

SAMPLE

Vincent ISOZ

Cookbook (M2)

2017-09-05 (V0.9 Revision 104), {oUUID 1.858}

Alphorm.com
DÉMOCRATISER LE SAVOIR

Table Of Contents

Table Of Contents **2**
Index Of Tables **15**
Introduction **16**
About the writers 18
ISOZ Vincent 18
KOLANI Daname 19
Warnings 20
How to use this book 21
Your opinion matters! 22
Useful Links 23
General 23
Forums 23
Non-MathWorks additional Toolboxes and Hardware 24
Open Data Sources 24
Others 25
Data Files 26
Donate 27
Bibliography 28
MATLAB Evolution **30**
MATLAB core (without Toolbox) **34**
New main core features (to be complete) 34
MATLAB 2014b (8.4) 34
MATLAB 2014a (8.3) 34
MATLAB 2013b (8.2) 34
MATLAB 2013a (8.1) 34
MATLAB 2012b (8.0) 35
MATLAB 2012a (7.14) 35
MATLAB 2011b (7.14) 35
MATLAB 2011a (7.12) 35
MATLAB 2010b (7.11) 35
MATLAB 2010a (7.10/2010) 35
MATLAB 2009b (7.9/2009) 36
MATLAB 2009a (7.9/2009) 36
MATLAB 2008b (7.8/2009) 36
MATLAB 2008a (7.6/2008) 36
MATLAB 2007b (7.5/2007) 37
MATLAB 2007a (7.4/2007) 37
MATLAB 2006b (7.3/2006) 37
MATLAB R14SP3 (7.1/2005) 38
MATLAB R14SP2 (7.0.4/2005) 38
MATLAB R14SP1 (7.0.1/2004) 39
MATLAB R14 (7.0/2004) 39
MATLAB R13 (6.5/2000) 39

SAMPLE

MATLAB R10 (5.2/1998) 39
MATLAB R9 (5.1/1997) 40
MATLAB R8 (5.0/1996) 40
Cite MATLAB 41
User Interface 41
Use Help 53
Edit a native function source code (if available) 55
Most asked user preferences 57
General commands 59
Execute system commands 59
Call R from MATLAB 61
Get MATLAB system information 63
Some dates stuff 69
Manage variables 70
Some Command Window controls 72
Use the startup.m file 74
Use the clipboard 77
Comments and multiple line commands 78
Save workspace/Use commands diary 79
Arithmetic operations on scalars 82
Operators of comparison 85
Real numbers manipulations 87
Command Windows real numbers formatting and output 89
Arithmetic operations on complex numbers 92
Identify the number type 95
Rounding results 97
Set (vectors) operations 98
Important practical cases 105
Counting/Removing NaN in cell arrays 105
Counting/Removing/Replacing NaN in (numeric) vectors 106
Finding the closest number in an vector 108
Arithmetic operations and manipulations of vectors 109
Important practical cases 118
Clean a vector 118
Build an orthonormal vector base 119
Arithmetic operations and manipulations of matrices 121
Important practical cases 140
Removing NaN in matrices (arrays) 140
Sampling matrix columns 141
Creation multiple dimensional matrices 142
Manipulating tables (structure array) 144
Mathematical functions 150
Statistical (core elementary) functions 163
Random variable generation 173
Use of constants (permanent variables) 178
Working with strings 181
Working with dates 188
Important practical cases 195
Convert year/month/day matrix as dates 195

SAMPLE

Difference in milliseconds/seconds between 2 dates 196
Simple user interactions 200
Message boxes 200
Input 201
Input Dialog 202
UI set color 205
UI save 206
UI open 206
UI get dir 208
Wait bar 209
Question dialog 212
Menu 213
List dialog 215
Send E-mail 216
Data management 217
MATLAB files (*.mat and *.m) 217
Manage working paths 220
Browse and manage folders and files 226
Import/Export csv files 233
Import very clean local csv files (with comas, without headers, without texts, without dates) 233
Import clean local csv files (with semi-colons, with headers, without text, without dates) 236
Import local csv files (with semi-colons, with headers, with text, with dates) 237
Merge csv files 240
Import web csv files (with semi-colons, with headers, with text, with dates) 242
Import from Yahoo Finance 243
Export to csv file 247
Import/Export xls/xlsx (Microsoft Office Excel) files 249
Import xls/xlsx local files 249
Export to xls/xlsx files 253
Import data with the wizard 255
Charts 262
Demo 262
Data sets 266
Chart wizard 268
Control chart window 299
Pick-up points on chart 310
X-Y plots 312
Analytical based plot 312
Vector based plot with lines and stems 313
Animated vector based plot 316
Vector based plot with various lines, colors and points 318
Vector based multiple plot with legends and titles 321
Horizontal/Vertical lines method 323
Stairs Plot 324
Vector based plot with error bars 325
Vector based semi-log X plot with custom tick space 326
Vector based semi-log Y plot 327
Vector based log-log plot with data cursor tip 328
Simple shapes (circles, rectangles) 330
Multiple plots (subplots, lattices) 332
Subplots *332*
Tab plots *335*

SAMPLE

Greeks letters in legends and titles (LaTeX style) 336
Plots with dates in axis 337
Multiple axis on line plots 339
Magnifier on line plot 342
Points density 343
2D vertical/horizontal bar charts 344
Multiple series bar chart *344*
Stacked bar chart *346*
Stacked normalized bar chart *347*
Bar chart with error bars *348*
Bar chart with bar having a special a color *349*
Tornado Chart *350*
Bar chart with line and multiple axis *351*
Two series bar chart with multiple axis *352*
Probability histogram with density curve (pdf) *354*
Handle charts 355
2D Correlation Heatmap 362
3D vertical bar charts 364
2D/3D sectors charts (cheese/pie charts) 366
2D Surface/Area charts 370
Radial (rose) bar chart/arrow (compass) chart 372
Pareto chart 375
Plotmatrix 376
3D parametric plots 377
Vector based 3D parametric line plot 377
Animated or non-animated analytic based 3D parametric line plot 378
Animated vector based 3D parametric line plot 379
Vector based 3D stem parametric line plot 380
Vector based 3D surface parametric plot 381
3D surface plots 392
ezsurf 392
ezsurfc 393
surf 394
surf with an explicit or implicit function! *402*
Animated surf 403
mesh 404
surfc 407
contour3 (3d contours) 408
surf with surf projection 411
contour (2d contours) 414
waterfall 418
2D Gradient (quiver) 419
3D Gradient 420
Perpendicular vectors 423
Surface plots of nonuniformly sampled data 424
Cubic (triangle base interpolation) *424*
Delaunay (triangular connections) for open surfaces *426*
Hull convolution for closed surfaces *427*
Spline (biharmonic-spline) *429*
Image mapping plots 430
Map an image on a 2D plane chart 430
Map an image on a 3D surface chart 431
Sphere plot with data mapping and animated rotation 433
Export animated plots as *.avi or *.gif 436

SAMPLE

Export to *.avi 436
Export to *.gif 437
Engineering/Simulations 438
Bouncing ball 438
Lagrange points potential surface 439
Finite Element Method 442
FEM Mesh simulation by simple average *442*
FEM Building 2D regular square grid *444*
FEM Building 3D regular square grid *444*
Vector filed equation analysis (electric field) 445
Vector field snapshots analysis 447
Regression/Interpolation/Extropolation 466
OLS univariate regression 466
Spline univariate interpolation 468
Spline multivariate interpolation 469
Fractals 471
Save chart as image 473
Save on drive 473
Copy in the clipboard 475
Analysis 476
Estimate value of an univariate polynomial on various points 476
Calculate the roots of an univariate polynomial 476
Find the local min/max of an univariate function in a range 479
Find the local min/max of a bivariate function 481
1st Method (very old one) 481
2nd Method (modern one) 483
Discrete convolution 483
Integration 486
Pseudo-symbolic integration of univariate polynomial 486
Numeric univariate integration 486
Numeric trapezoidal univariate integration 487
Double numeric integration 488
Derivation 489
Pseudo-symbolic derivation of polynomial 489
Numeric derivation 490
Differential equations (ODE/PDE) 493
Pendulum (ODE) 493
Lotka-Volterra (system of ODE) 496
Lorenz attractor (system of ODE) 498
2 body motion (system of ODE) 500
Heat equation (parapolic PDE) 504
Fast Fourier Transform (DFT) 508
Image processing 512
Get image properties 512
Show an image properly 514
Decompose images alpha layers 515
Build an image 516
Plot some basic images manipulations and save image 517
Filter on luminosity value of a given channel 518
Draw on an image and export the result 519
Compare images 522
Add (superpose) images 524

SAMPLE

2D FFT 527
Video processing 528
Get video properties 528
Play video 529
Extract a frame 530
Export all video frames in a folder 531
Build a video from a list of (pre-processed or not) images 532
Sound processing 533
Get sound properties 533
Play, Pause (resume), Resume, Stop sound playing 534
Plot a sound spectrum 535
Sound FFT (DFT) 537
Generate a sound and save it 539
Play sounds in sequence or parallel 540
Scripting 541
Script Editor most asked preferences 541
Well commented script 543
Define help text 545
Use ToDo, FixMe and Notes 547
Autocompletion 549
Conditional structure (if/switch) 550
if statement 550
switch statement (with if...) 552
Iteration structures 554
For...End iteration structure (and optimization) 554
While...End iteration structure 557
Functions (functions files) 559
One input, One output (OIOO) 559
Multiple inputs, One output (MIOO) 561
One input, Multiple outputs (OIMO) 562
Multiple inputs, Multiple outputs (MIMO) 564
Optional arguments 565
Variables scope 566
Global vs Local variables *566*
Persistent variables *569*
Anonymous function as input, chart output 570
Recursive functions 572
Local functions (multiple functions in one script) 574
Profile a script 575
Catching errors 579
Multiple callable functions in one *.m file 579
Debugging technics 582
M-lint 583
M-lint real time analyzer *583*
M-lint reports *585*
Echo on/Echo off 589
For simple scripts *589*
For functions scripts *593*
Manipulations during a run 594
Debugging (db...) functions 596
Add conditional breakpoints 603
Create a more complex wait bar 605

SAMPLE

Import an XML file 606
Export an XML file 609
Automation 611
Generate a Microsoft Word report 611
Put values into Microsoft Excel 613
Get values from a Microsoft Excel VBA function 616
Create a ToolBox (App) 618
Install/Uninstall a Toolbox 624
Publish a script (documented script) 628
Check Readiness 634
Optimization tricks 636
Statistically check code efficiency 636
linspace vs sequence 638
Simple arithmetic operation vs bsxfun 639
Create userforms (GUI) 641
Create userforms using GUIDE 641
Edit existing GUI 650
Manage static texts (legends) 651
Manage buttons and associate a callback 656
Create axes area for figures, images, ... 658
Manage check-boxes and associate a callback 663
Manage radio buttons group 666
Manage pop-up menu and get selected value 672
Manage multiple selection List box 677
Manage sliders 683
Manage tables 686
Userform calling another userform 695
Write/Get GUI variables in MATLAB workspace 697
Create userforms using only script 700
Important practical cases 701
Create a GUI that opens a file 701
Object Oriented Programming (OOP) 708
MVC (Model-Vue-Controller) 708
Symbolic calculations (Symbolic Toolbox) 710
New main core features (to be complete) 710
MATLAB 2014a (8.3) 710
MATLAB 2013b (8.2) 710
MATLAB 2013a (8.1) 710
MATLAB 2012b (8.0) 710
Series 711
Limits 712
Derivate/Partial Derivate 713
Integration 714
Taylor series 715
Heaviside function 717
Matrix determinant 718
Integral Transformations 719
Fourier Transform 719
Laplace Transform 721
Z Transform 722
Jacobien 723
MuPAD 724

SAMPLE

Databases (with Database/Datafeed toolbox) 725
Connect to mySQL 5.5.36 725
Connect to Microsoft Access 2003/2010 (*.mdb/*.accdb) 730
Connect to Oracle Express 11g 734
Connect to MS SQL Server Express 2008 738
Connect to Yahoo! Finance 744
Statistics (with Statistical and Curve Fitting Toolbox) 745
Datasets 745
Import/Export *.csv files as data set (data frame) 745
Import *.xls/*.xlsx files as data set (data frame) 748
Convert structure array to data sets 749
Clean messy or missing data in datasets 751
Select/Remove column (variable) 753
Create subsets and filter 753
Delete rows and clean duplicates 755
Add variables 755
Change variable name 756
Change column order 756
Sort a dataset 757
Merge datasets 758
Dataset tabulations and crosstab analysis 759
Random sample of a dataset 761
Simple random sampling (equal probabilities) 761
Balanced Stratified random sampling (unequal probabilities) 762
Stratified random sampling (unequal probabilities) 765
Plots 766
Histogram/CDF/PDF 766
Box-Whisker plot 769
Scatter Group plot 770
QQ plot 771
Histogram Scatter 772
Copulas 773
Gaussian copula 773
Distributions 776
Descriptive statistics 778
Geometric mean 778
Harmonic mean 779
Trimmed mean (M-statistics) 780
CDF/iCDF/Quantile/Centile 781
Skewness/Kurtosis 782
Hypothesis tests 785
Tests power, sample size and resolution 785
Power of a Z test 785
Sample size of a one sample Z test 786
Difference (resolution) of a one sample Z test 786
Power of a t test 787
Sample size of a one sample t test 787
Sample size of a one sample p test 788
One sample Z-test 788
One sample t-test 789

SAMPLE

One sample t-test with explicit target mean 790
Two sample homoscedastic t-test 791
Two sample heteroscedastic t-test (Welch test) 792
Paired t-test 793
Chi-square variance test 794
Exact Fisher Test 795
Two sample F-test for equality of variances 796
Two sample Levene's test for equality of variances 797
Brown-Forsythe (median) 797
Absolute Levene 799
Quadratic Levene (square) 800
Test for the median (one sample sign test) 801
Binomial signed rank test (dixit: median test paired samples) 802
Two unpaired samples Wilcoxon (Mann-Withney) rank sum test 803
Wilcoxon signed rank test for 1 sample 804
Wilcoxon signed rank test for two paired samples 805
ANOVA 806
One-way fixed factor canonical ANOVA 806
Two-way fixed factors canonical ANOVA 809
Friedman's test (canonical non-parametric ANOVA based on ranks) 811
Kruskal-Wallis test (canonical non-parametric one-way ANOVA 813
Adequation tests 816
Chi-square goodness of fit for Poisson distribution 816
Kolmogorov-Smirnov (one sample) adequation test 817
Anderson-Darling adequation test 819
Regression Techniques 820
Ordinary Least Squares (parametric) univariate regression 820
1st method (coefficient and constant cannot be extracted) 820
2nd method (coefficient and constant and others can be extracted) 824
3rd method (using Curve Fitting Toolbox) 826
Manage outliers 832
Residuals and Leverage (residual diagnosis) 836
Regression diagnostics wizard 837
Ordinary Least Squares (parametric) multivariate regression (with/without interactions) 839
1st method (coefficient and constant cannot be extracted) 839
2nd method (coefficient and constant and others can be extracted) 842
Polynomial multivariate regression using OLS and Curve fit Toolbox 844
(Forward) Stepwise regression 847
General Linear Models (GLM) 850
Normal regression 850
Log-linear Poisson regression 851
Binomial (logistic) regression 852
Canonical Correlation 855
Akaike Information Criterion (AIC) and Bayesian Information Criterion 859
Data Mining/Machine Learning 861
Naive Bayes 861
k-NN (k-Nearest Neighbors) 866
K-means (Euclidian full) 869
K-medoids 876
Hierarchical Ascendant Classification (HAC) dendrogram 878

SAMPLE

CART Classification trees (Gini index based) 884
Non classical multidimensional scaling 891
Parametric Principal Component Analysis 893
Linear Discriminant Analysis (Fisher Discriminant) 899
Quadratic Discriminant Analysis 908
Control charts (CC) 910
P attribute CC 912
NP attribute CC 914
C attribute CC 915
U attribute CC 917
Xbar-R continuous CC 919
Rbar-R continuous CC 920
Sbar-S continuous CC 922
I-EM continuous CC 923
Moving Average continuous CC 926
Hotelling's bivariate ellipse control chart 928
Design of experiments 932
Two-level full factorial design 932
Full factorial design 933
Fractional factorial design 934
Simulations techniques 935
Randtool wizard 935
Kernel Smoothing 936
Monte Carlo standard technic 938
Monte Carlo with acceptance reject 940
Monte Carlo antithetic variables technique 941
Monte Carlo with antithetic quasi-random variables technique 943
Monte Carlo by control variable (variance reduction technique) 945
Monte Carlo importance sampling 947
Latin HyperCube 950
Monte Carlo Markov Chain Gibbs Sampling 951
Optimization (with Optimization Toolbox/Global Optimization Toolbox) 953
Linear optimization using LP Simplex (LPS) 953
LP-S Factory optimization problem (inequality linear system) 953
LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (equality linear system without target value) 957
LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (equality linear system with target value) 959
Function constrained optimization (FCO) 961
FCO with constraints and without specific target 961
FCO with constraints and specific target 963
FCO with nonlinear constraints (and specific target) 966
Quadratic optimization (QO) 968
QO using conjugate gradient method (Q-CG) 968
QO using interior point convex method (Q-IPC) 972
Genetic algorithms (GA) 975
Simple close-form unconstrained function without upper and lower bounds 975
Neural Network GA optimization for DOE 978

SAMPLE

Finance (with Finance/Optimization and Econometrics Toolboxes) 983
Plots 983
OHLC Chart (Open-High-Low-Close) 983
Fractional prices 984
Return 985
Maximum Draw Down 986
Box-Cox transformation 988
Simple capital calculations 990
Composed Interest (fixed future value) 990
Composed Interest (fixed present value) 990
Spot rate (zero coupon rate) to forward rate 991
Amortization and deprecation 993
Straight line deprecation (SLN) 993
Sum of years digit (SYD) 993
Fixed declining-balance depreciation schedule (DB) 994
Double declining balance (DDB) 994
Loans and credits 995
Payment for a loan based on constant payments and a constant interest rate (PMT) 995
Principal Payment (PPMT) 997
Interest Payment (IPMT) 998
Payment Present Value (PV) 999
Plot resume of loan payment 1000
Annuities 1002
Postnumerando actual annuity value (PV) 1002
Postnumerando forward annuity value (PV) 1002
Postnumerando number of periods (NPER) 1003
Postnumerando annuity rate (ANNURATE) 1003
Corporate finance 1004
Net Present Value (NPV) 1004
Periodic NPV 1004
non-Periodic NPV 1004
Internal Rate of Return (IRR) 1005
Periodic IRR 1005
non-Periodic IRR 1005
Modified Internal Rate of Return (MIRR) 1006
Treasury bills/Securities (bonds) calculations 1007
Clean price of a security (BNDPRICE) 1007
Price of treasury bond/security (TBILLPRICE) 1008
Yield of a treasury bond (TBILLYIELD) 1009
Discount rate of a security (DISC) 1010
Annual yield for a discounted security (YIELDDISC) 1010
Bond duration, Macaulay duration and convexity 1011
Naive Markowitz portfolio models 1013
Optimal short-long portfolio (without transaction costs) 1013
Optimal short-long tangent portfolio (without transaction costs) 1015
Optimal portfolio with implicit constraints no short sales allowed (without transaction costs) 1016
Optimal portfolio with implicit constraints no short sales allowed (with transaction costs) 1020

SAMPLE

Get portfolios on the efficient frontier with explicit constraints (without transaction costs) ... 1021
Equally weighted portfolio (EWP) (without transaction costs) ... 1023
Maximum Sharpe Ratio (max. diversification) portfolio with implicit constraints (without transaction costs) ... 1025
Optimal portfolios for target returns and CVaR minimization (without transaction costs) ... 1027
Optimal portfolios for target risks and CVaR minimization (without transaction costs) ... 1028
Value at risk (VaR) ... 1029
Normal univariate absolute VaR ... 1029
Conditional (absolute) Normal VaR ... 1030
Historical (absolute and relative) non-parametric VaR ... 1030
Monte Carlo relative Multivariate Normal VaR ... 1031
Options Stocks/Futures Pricing & Monitoring ... 1033
Pricing European Call & Put paying or not dividends (Black & Scholes-Merton Model) ... 1033
Pricing Future Call (Black's Model) ... 1035
Greeks on European Call & Put paying or not dividends ... 1035
European Option (call/put) Delta ... 1035
European Option (call/put) Gamma ... 1036
European Option (call/put) Vega ... 1036
European Option (call/put) Lambda ... 1036
European Option (call/put) theta ... 1037
European Option (call/put) Rho ... 1037
Options trading strategies (payoff diagrams) ... 1037
Implied volatility surface of an European option ... 1038
Volatility smile of an European option ... 1040
CRR binomial European option pricing model ... 1044
Time series ... 1046
Runs test (Wald-Wolfowitz) ... 1046
Autocorrelation ... 1048
Partial autocorrelation ... 1050
Moving average MM3 (without ARIMA) ... 1052
Simple Exponential Weighted Moving Average (EWMA) ... 1054
Univariate AR(2)/ARIMA(2,0,0) ... 1057
Univariate ARIMA(0,0,0)/Moving Average ... 1060
Univariate ARIMA(0,1,0) ... 1063
Univariate ARIMA(0,1,1)/Simple Exponential Smoothing (EWMA) ... 1067
Univariate ARIMA(0,2,2)/Holt's double exponential smoothing ... 1069
Univariate ARMA(1,1)/ARIMA(1,0,1) ... 1072
Univariate GARCH(1,1) ... 1075
Applications (without Financial Toolbox) ... 1080
Bond portfolio with duration/convexity targets ... 1080
Bond portfolio with duration/convexity targets and maximization of average yield ... 1081
Backtesting long position buying-holding and selling SPY ... 1083
Brownian motion and Wiener Process ... 1090
Risk-return asset pricing models ... 1092
Capital Asset Pricing Model (CAPM) ... 1092
Three-factor Fama-French model (FFM) ... 1093

SAMPLE

Fama-MacBeth 1095
Options Pricing 1098
CRR binomial option pricing model without Financial Toolbox 1098
Drifted CRR binomial option pricing model without Financial Toolbox 1099
Jarrow-Rudd binomial option pricing model without Financial Toolbox 1101
Jarrow-Rudd binomial risk neutral option pricing model without Financial Toolbox 1102
Leisen-Reimer binomial option pricing model without Financial Toolbox 1104
Tian binomial option pricing model without Financial Toolbox 1106
Portfolios models 1108
Global Minimum Variance Efficient Portfolios (GMVP) without Toolboxes using langrangien multiplier 1108
Global Minimum Variance Efficient Portfolios (GMVP) using only Optimization Toolbox 1109
Portfolio Performance 1118
Michaud Optimal Portfolio (bootstrap/resample efficiency) 1120
CVaR portfolio 1124
Black-Littermann portfolio 1131
Signal processing (Signal processing & Communications System Toolbox) 1135
Plot discrete signal functions 1135
Found the peaks of signal 1138
Add echo to a sound (1D digital filter) 1139
Hamming Error Correcting Code 1140
Circuit Analysis (Simulink Toolbox) 1141
Create a simple signal filter 1148
Create a simple manual switch on/switch off signal 1151
Create an RLC branch circuit 1155
1. Over-damped circuit 1159
2. Critically over-damped circuit 1160
3. Under-damped circuit 1162
Fuzzy Logic (Fuzzy Toolbox) 1165
Neural Networks (Neural Network Toolbox) 1168
Image Processing (with Image Processing and Acquisition+Statistical Toolboxes) 1180
Play with the image tool wizard 1180
Convert image to black and white 1184
Rotate image 1185
Resize an image 1186
Image as a function 1187
Image Isophobes 1189
Image histogram 1190
Filters (blur, motion) on an image 1195
Partially blur an image 1197
Crop an image 1200
Region of interest 1201
Convert a *.gif file into an *.avi 1202
Overlay images with masks 1205
Compare Images in colormapping 1209
Sliding Neighborhood Operations 1210

SAMPLE

MRI (Magnetic Resonance Imaging) Dataset ... 1212
Objects recognition and extraction ... 1221
Extraction without reference images ... 1221
Extraction with reference shape matrix ... 1222
Extraction with reference image ... 1223
Image filters (convolution) ... 1229
Gaussian (blur) filter ... 1229
1st method using imfilter() ... 1229
2nd method using convn() ... 1231
3rd method using conv2() ... 1232
External Camera Acquisition ... 1233
Bioinformatic (with Bioinformatics and Statistics Toolbox) ... 1239
Complete missing values ... 1239
Access sequence databases ... 1241
Shortest Path ... 1246
Clustermap ... 1247
Parallel Computing (with Parallel Computing Toolbox/PCT) ... 1252
Clusters computing ... 1252
NVidia GPU computing ... 1257
Compiler Toolbox (C#, C++, Compiler Toolbox) ... 1260
Create a *.p (protected script) file (aka "pcode") ... 1260
C MEX file ... 1261
Create executable (*.exe) ... 1264
Call MATLAB from Microsoft Excel ... 1269
Use the *.dll and *.xla on another computer ... 1287
Call MATLAB from C# (VS Express 2010) ... 1287
Call MATLAB from C++ (VS Express 2010 x64) ... 1293
Compile DLL and call from C# ... 1319
Passing a scalar ... 1319
Passing a constant, vector and matrix ... 1329
Compile DLL and call from C ... 1331
Compile JAR and call from JAVA ... 1335
VHDL ... 1356

SAMPLE

Index Of Tables

Table 1 Standardized color codes ... 307
Table 2 Common dates format syntaxes ... 338
Table 3 ODE solvers ... 494

Introduction

MATLAB is the abbreviation for ***MA**trix **LAB**oratory* and is actually the leading environment for technical computing. Originally written in Fortran, by C Moler, MATLAB was designed to facilitate the access to the matrix framework developed for the LINPACK and EISPACK projects. The current version, written in C by the MathWorks Inc., exist in a Professional version and a Student version. Its availability is ensured on multiple platforms: Linux, Mac, Windows and several parallel machines.

MATLAB is a comprehensive and almost easy to use for scientific computing but very expensive for students or independent/small companies (unlike R, Octave, Scilab or other free open source equivalent alternatives). It provides to engineers, researchers and any scientific an interactive system that integrates a lot of scientific numerical computation and visualization algorithms (but can do much better!). It is a powerful, almost open and programmable system allowing especially remarkable gains in productivity and creativity.

MATLAB is relatively complete, open and extensible for calculation and visualization algorithms. It has hundreds (if not thousands, depending on the version and the optional toolbox around the MATLAB core solution) mathematical, scientific and technical functions. The matrix approach allows MATLAB to process data without large size variables and perform numeric and symbolic calculations quickly and reliably. Thanks to the graphics capabilities of MATLAB, it becomes very easy to interactively change the various graphics settings and to adapt them according to our wishes.

The fact that the source code of some functions integrated in MATLAB can be inspected gives sometimes the engineers to develop better custom made solutions (also do Python Scipy...).

MATLAB has its own language that is very fast and allows time savings for Engineers compared to other languages like C, Fortran, Python and the TurboPascal even if the language is horrible to use (non-typed, bad closed lined, etc.). With MATLAB, we can make connections dynamically, to C, .Net or Fortran programs, exchange data with other applications (via DDE: MATLAB server or client) or use MATLAB as analysis and visualization engine.

In terms of development speed, MATLAB is always faster than C/C++. This is to be expected with a higher-level language, just as C/C++ is faster to develop with than assembler.

But a C/C++ programmer with an infinite amount of development time can beat MATLABs speed, in the same way an assembly programmer with an infinite time can beat a C/C++ programmer.

MATLAB also includes a set of tools specific to engineering areas called Toolboxes. Essential for most users, Toolboxes are collections of functions that extend the MATLAB environment to solve specific types of problems. The areas covered are varied and include signal processing, automatic control, system identification, neural networks, fuzzy logic, structure calculation, statistics, database access, etc.

This document can also strongly help you to prepare the following certifications:





SAMPLE

Notes:

N1. The first exercises are almost the same as those made in the theoretical statistics courses, statistical process control, decision analysis with Microsoft Excel, Minitab, SPSS and MATLAB or during the engineering project management course (in which the calculations are made by hand).

N2. In this book we show normally only tools using mathematical concepts for which we studied the detailed proof (demonstrations available in my book on Applied Mathematics) in our courses and are used as part of our consulting activities. If this is not the case, the topic title is followed by the abbreviation WP that stands for: Without Proof.

N3. I wrote this document only for fun to validate the results obtained by hand and especially to play with mathematical theorems I studied in detail as part of my studies (mathematical proofs available in my book on Applied Mathematics).

N4. Following questions from readers: No, I am not paid by The MathWorks to do their Ad ... software like:

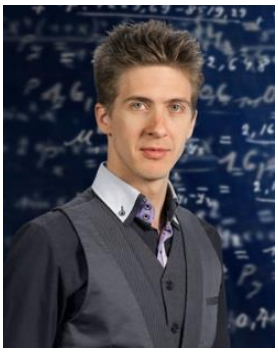
Minitab, XLStat SPSS, R (see my PDF about this program), SAS, PSPP, SyStat, Gauss, Matlab, Statistica, Stata, MedCalc, StatsDirect, SigmaXL, NumXL, JMP, Weibull ++, Design-Expert, PlanExpert, UNISTAT, QIMacros (Quality Improvement macros), EasyFitXL, ...

do almost the same relation about topics covered in this document or just specific topics. I just had to make a choice... (I cannot spend my time writing books on all types of software!) And I focused on the software used by the majority of my customers working in finance and insurance and R&D. However if enthusiasts people want to reproduce the contents of this book with their favorite statistical software they do not have to hesitate! It would even be interesting to compare the results!! If I have time I will write the same content but with SPSS.

SAMPLE

About the writers

ISOZ Vincent



Resident until this date in Lausanne (Switzerland)

Trainer: Physicist Engineer HES (B.Sc.)

Year of birth: 1978

E-mail: isoz@sciences.ch

Employee at Scientific Evolution Sàrl specialized in consulting and training for high-potentials especially quantitative analysts in project management, quality, engineering, finance and decision making. Incidentally, happy freelancer that gathers since young information from articles everywhere and now addicted to everything related to applied mathematics in the management of organization (Businesses, States), TQM (Total Quality Management) with Post-Taylorism and related softwares.

Consultant in applied mathematics tutoring quantitative analysts (graduate and postgraduate) in French, English and German and author of several French and English eBooks in the following areas:

- statistical process control (parametric and nonparametric methods)
- predictive modeling / advanced decision (decision trees, Markov chains)
- Operations Research (quasi-Newton, simplex, genetic algorithms, algorithm GRG)
- data mining (neural networks, PCA, CA, regressions, scoring, clustering, etc.)
- risk modeling in project management and corporate finance (monte carlo, bootstrapping)
- project management (best practices and theoretical models EFQM+Six Sigma, MS Office Project)
- ISO 9001:2008, 5807:1985, 10015:1999, 31000+31010:2009, 8258:1991, 10017:2003
- Adobe Photoshop and Illustrator
- 12 applications of the Microsoft Office System (Project, Visio, SharePoint, Access, etc.)

At this date I made interventions in over 200 companies, including some of to the Fortune 500 according to the 2009 listing and also in three universities and engineering schools in Switzerland for databases and stochastic risk simulations trainings. I also gave training in several multinational to executives in one to one. Sometimes I give training on softwares like MS Project, MS Visio, MS Access and approximately 20 others softwares that I delegated the training organization to companies specialized in computer softwares trainings (university degree level and below).

Finally, I would advise anyone wishing to really master the subject to read my e-book on Applied Mathematics (~ 5'400 pages).

SAMPLE

KOLANI Daname



I'm currently living in Casablanca (Maroc)

Graduate in Finance from the Business School ENCG-Casablanca

Born in 1990, I am from Togo.

E-Mail: daname.kolani@hotmail.com

I am passionate in the field of finance, economics, and other corollary as econometrics, also in statistics & probabilities. Apart, the theories associated with those fields, I have a great interest in tools that underpin all of these areas. I am bilingual (French & English) and my specific competences include:

- Corporate Finance (Investment decision & Financing, Financial Analysis and Diagnosis, Management Control and Company Evaluation...)

- Market finance (financial portfolio management, financial asset valuation, risk management tools ...)

I also have an interest in computational finance skills in R Project, VBA, Excel, MATLAB, EViews & SPSS.

I aspire to get into consulting and thereby contribute to the optimization of financial decisions and the implementation of efficient financial strategy consistent with the business model of companies.

SAMPLE

Warnings

The content of this document is elaborated by a development process by which volunteers reach a consensus. This process that brings together volunteers, research also the point of view of people interested in the topic of this document. The person in charge of this document administers the process and establishes rules to promote fairness in the consensus approach. It is also responsible for drafting the text, sometime for testing/evaluating or independently verifying the accuracy or completeness of the presented information.

We decline no responsibility for any injury, damage or any other kind, special, incidental, consequential or compensatory, arising from the publication, application or reliance on the content of this document. We make no express or implied warranty on the accuracy or completeness of any information published in this document, and do not guarantee that the information contained in this document meet any specific need or goal of the reader. We do not guarantee the performance of products or services of one manufacturer or vendor solely by virtue of the document content.

By publishing texts, it is not the intention of this document to provide services on behalf of any person or entity or performing any task to be accomplished by any person or entity for the benefit of a third party. Anyone using this document should rely on its own independent judgment or, where that is appropriate, seek the advice of a qualified expert to determine how to exercise reasonable care under all circumstances. The information and standards on the topic covered by this document may be available from other sources that the reader may wish to visit in search of points of view or additional information not covered by the contents of this document.

We have no power in order to enforce compliance with the contents of this document, and we do not undertake to monitor or enforce such compliance. We have no certification, testing or inspection activity of products, designs or installations for safety or health of persons and property. Any certification or other statement of compliance regarding information relating to health or safety of persons and property, mentioned in this document, cannot possibly be attributed to the content of this document and remains under the responsibility of the certification center or the concerned reporter.

SAMPLE

How to use this book

At the university level, this book can be used for a PhD, graduate level or advanced undergraduate level seminars in many exact and pure sciences fields. The seminars where we use this material is part of Scientific Evolution Sàrl program, where the trainees typically already have taken undergraduate or graduate courses in their respective specialization.

Because the methods of applied mathematics are learned by practice and experience, we view a seminar on MATLAB as a learning-by-doing seminar. We structure our mathematical modeling seminars around a set of problems that require the trainee to construct models that help with planning and decision making. The imperative is that the models should be consistent with the theory and back-tested. To fulfill this imperative, is necessary for the trainee to combine mathematical theory with modeling. The result is that the trainee learns the theory, and more importantly, learns how that theory is applied and combined in the real world. The ability to criticize and identify limitations of dangerous mathematical tools is the most valuable feature of our seminars.

The problems with solutions in the text provide the opportunity to apply the text material to a comprehensive set of fairly realistic situations. By the end of the seminars the trainees will have enhanced their skills and knowledge of the most important theoretical and computing tools. These are valuable skills that are in demand by the businesses at the highest levels.

It is very difficult to cover all the material in this book in a semester. It takes a lot of time to explain the concepts to the trainees. The reader is encouraged to pick and choose which topics will be covered during the term. It is not necessary strictly necessary to cover them in sequence but it can help in a significate way…

In a nutshell, this book offers you a wide variety of topics that are amenable to modeling. All are practical.

SAMPLE

Your opinion matters!

As a reader of this document, you are the critic and the most important commentator. Your opinion counts and it is very interesting for us to know what is good, what can be better and topics that you would like to see be treated.

You can send me an e-mail to share what you liked or disliked in this document to ensure continuous improvement.

If you want to complete this support with a subject that is important to you and for which you have the mathematical proof please do not hesitate to contact me. I will integrate the subject specifying your name and surname.

Note that unfortunately I cannot answer to technical engineering questions or business issues for free by email for obvious professional reasons.

E-mail: isoz@sciences.ch

With that, I want to say a huge "THANK YOU" to all of my customers, partners, colleagues, students and friends who with their feedback, return on experience helped me write this e-book.

SAMPLE

Useful Links

General

http://ww.sciences.ch (the companion website containing mathematical proofs)

http://ww.google.com

http://www.youtube.com

http://www.scribd.com

http://www.mathworks.com/products/

http://www.mathworks.ch/ch/help/matlab/release-notes.html

http://www.mathworks.com/matlabcentral/fileexchange/

http://www.mathworks.ch/discovery/

http://www.mathworks.ch/discovery/gallery.html

http://www.mathworks.com/matlabcentral/newsreader/

http://www.undocumentedmatlab.com

http://www.imageprocessingblog.com

http://www.alphorm.com (for people that want the fundamentals part of this eBook as video):



Forums

http://ww.mathworks.ch/matlabcentral/answers/

http://www.talkstats.com

http://matlab.developpez.com

SAMPLE

Non-MathWorks additional Toolboxes and Hardware

http://www.mosek.com (optimization toolbox: non-free)

http://www.cvxr.com (another convex optimization toolbox)

http://www.asu.cas.cz/~bezdek/vyzkum/rotating_3d_globe/ (earth globe statistics: free)

http://www.nag.co.uk/numeric/MB/start.asp (high speed calculation toolbox: non-free)

http://psychtoolbox.org (world reference toolbox for psychologists)

http://ww.idquantique.com/random-number-generators/products.html (quantum rd generator)

http://www.tadeveloper.com (trading strategy technical analysis tool)

http://rtime.felk.cvut.cz/scheduling-toolbox (scheduling optimization toolbox)

http://www.bull.com French provider of HPC servers

http://www.techilatechnologies.com (distributed calculation solution)

Open Data Sources

https://opendata.swiss

http://ec.europa.eu/eurostat/data/database

http://www.quandl.com

https://data.gov.uk

http://www.data.gov

http://www.census.gov/data.html

http://www.opendatanetwork.com

http://open-data.europa.eu/en/data/

http://wiki.dbpedia.org/

http://dataportals.org/

http://www.healthdata.gov/

https://azure.microsoft.com/en-us/marketplace/

http://www.gapminder.org/data/

http://finance.yahoo.com/

SAMPLE

http://archive.ics.uci.edu/ml/index.html

http://www.ncdc.noaa.gov/data-access/quick-links#loc-clim

http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

Others

On the aberration of managing big business with spreadsheets softwares:

http://staffweb.cms.gre.ac.uk/~cd02/EUSPRIG/

SAMPLE

Data Files

The data files used in this book, that is to say:

- 213 Files (*.mat, *.csv, *.jpg, *.mov, etc.)
- Classified in 33 Folders
- For a total of 3.02 GB of data

are given with free perpetual updates only to the people that follows our corresponding MATLAB training or to people that pay (or donate) an equivalent amount of **499$**.

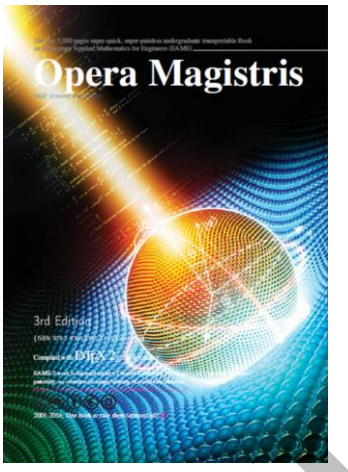
Universities, Engineering schools and Corporations need to buy only one time for a unlimited use of their students or employees but restricted to only one geographical localization.

For more information on the purchasing process, you can contact us by email at the following address: isoz@sciences.ch

SAMPLE

Donate

This MATLAB book belongs to the companion books of the 6,000 pages theoretical compendium *Opera Magistris* that is also a free book:



and also of the equivalent free books on R and Minitab.

In order to continue the writing of this compendium and to continue to evolve this free book on MATLAB your donations are welcome and we thank you in advance.

You can donate through Patreon:

https://www.patreon.com/sciences

or PayPal:

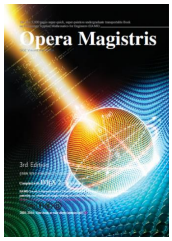
http://www.sciences.ch/htmlen/donate.php

SAMPLE

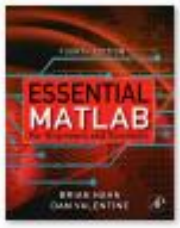
Bibliography

Here is a list of books of extraordinary educational and rigor quality I had the chance to have in my hands and that I strongly recommend the acquisition. I've read many others (more than a hundred!) but these latter are so bad they do not need to be mentioned.

The reader will understand that I recommend strongly to complete the reading of this e-book (non-exhaustive about MATLAB) by the book-list below.



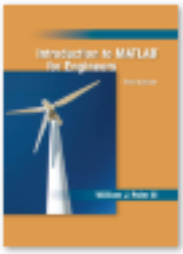
Opera Magistris / ~5'576 pages / Editions Sciences.ch / Vincent ISOZ / 3rd édition
ISBN: 978283999327



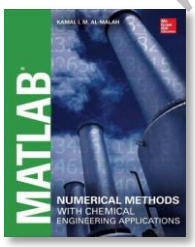
Essentials MATLAB for Engineers and Scientists / 427 pages / ELSEVIER Editions / Brian HAHN & Dan VALENTINE
ISBN: 9780123748836



MATLAB a practical introduction to programming and problem solving / 524 pages / ELSEVIER Editions / Stormy ATTAWAY
ISBN: 9780123850812

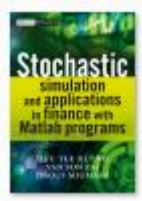


Introduction to MATLAB for Engineers / 577 pages / McGraw-Hill Editions / William J. PALM III
ISBN: 9780073534879



Numerical Methods with Chemical Engineering Applications / 419 pages / McGraw-Hill Editions / Kamal I. M. AL-MALAH
ISBN: 9780071831291

SAMPLE



Stochastic simulations and applications in finance with MATLAB programs / 356 pages / Wiley Finance Edition / Huu Tue HUYNH, Van SON LAI, Issouf SOUMARÉ

ISBN: 9780470725382



Financial Instruments Toolbox / 2347 pages / MathWorks Edition /

ISBN: -



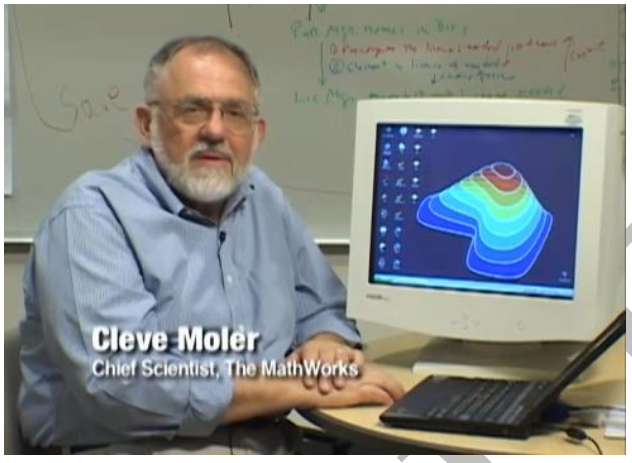
Undocumented Secrets of MATLAB-Java Programming / 701 pages / Chapman and Hall/CRC Edition / Yair M. Altman

ISBN: 1439869030

SAMPLE

MATLAB Evolution

MATLAB was programmed by Cleve Moler as an interface for EISPACK & LINPACK.



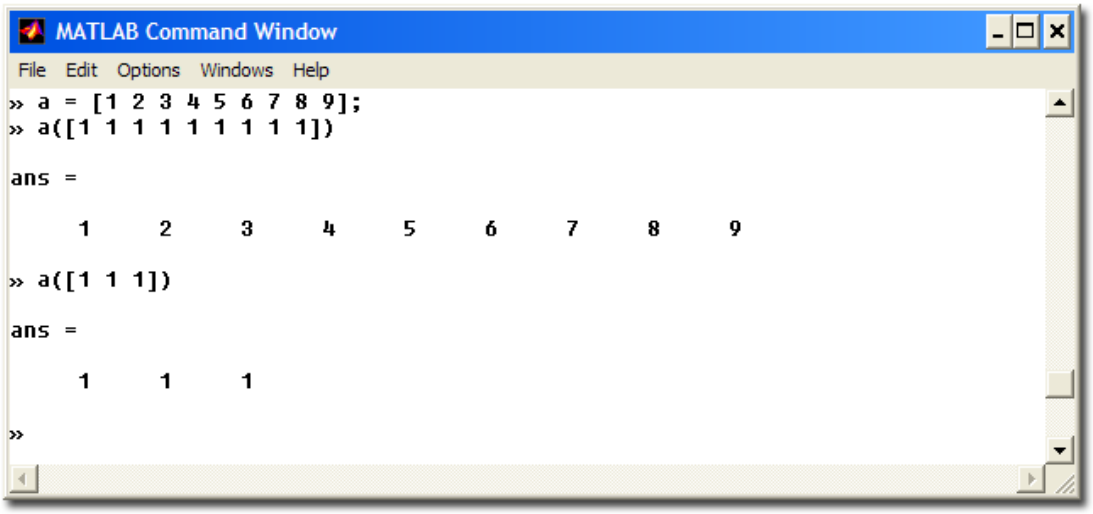
- 1957: Moler goes to Caltech. Studies numerical Analysis
- 1961: Goes to Stanford. Works with G. Forsythe on Laplacian eigenvalues.
- 1977: 1st edition of MATLAB; 2000 lines of Fortran 80 functions (now more than 8'000 functions)
- 1979: Met with Jack Little in Stanford. Started working on porting it to C
- 1984: *The MathWorks Inc.* is founded

Below is a screen shot from MATLAB 3.13:

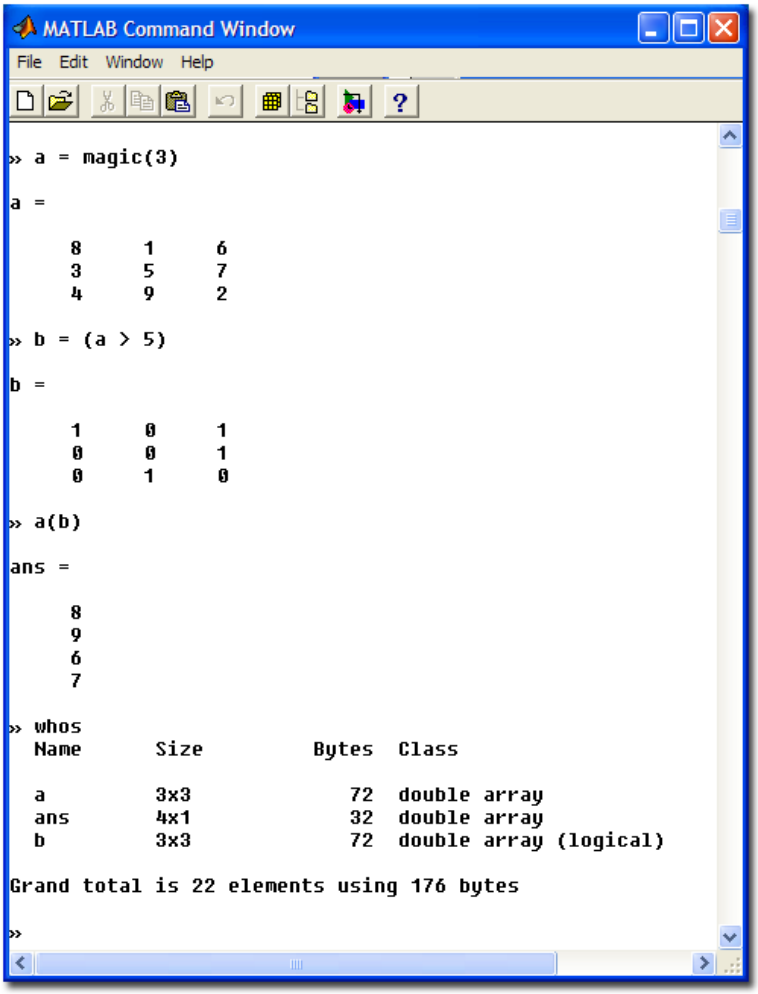


MPLE

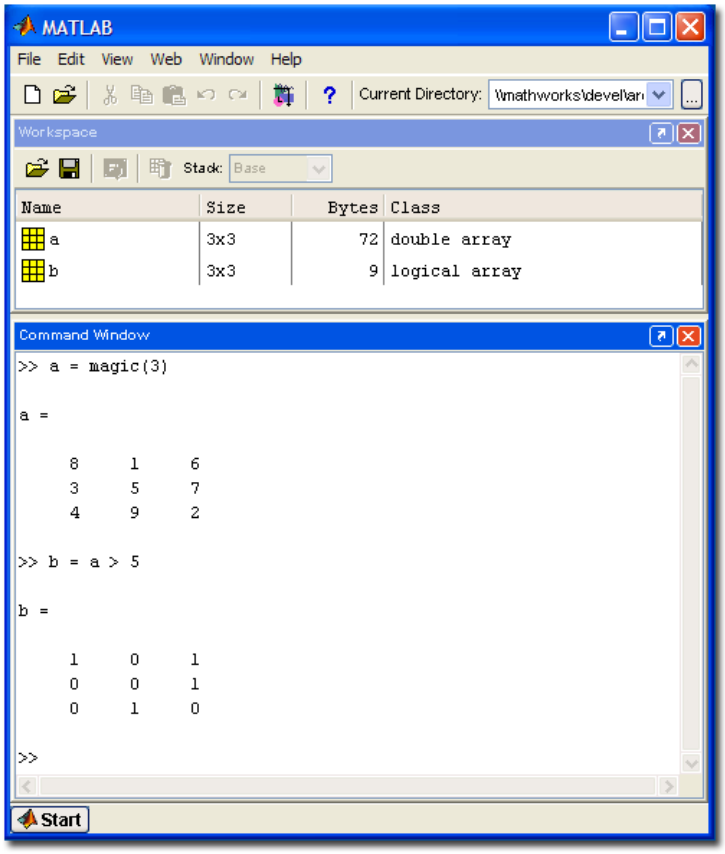
Here's a screen shot of MATLAB 4.2c (1994):



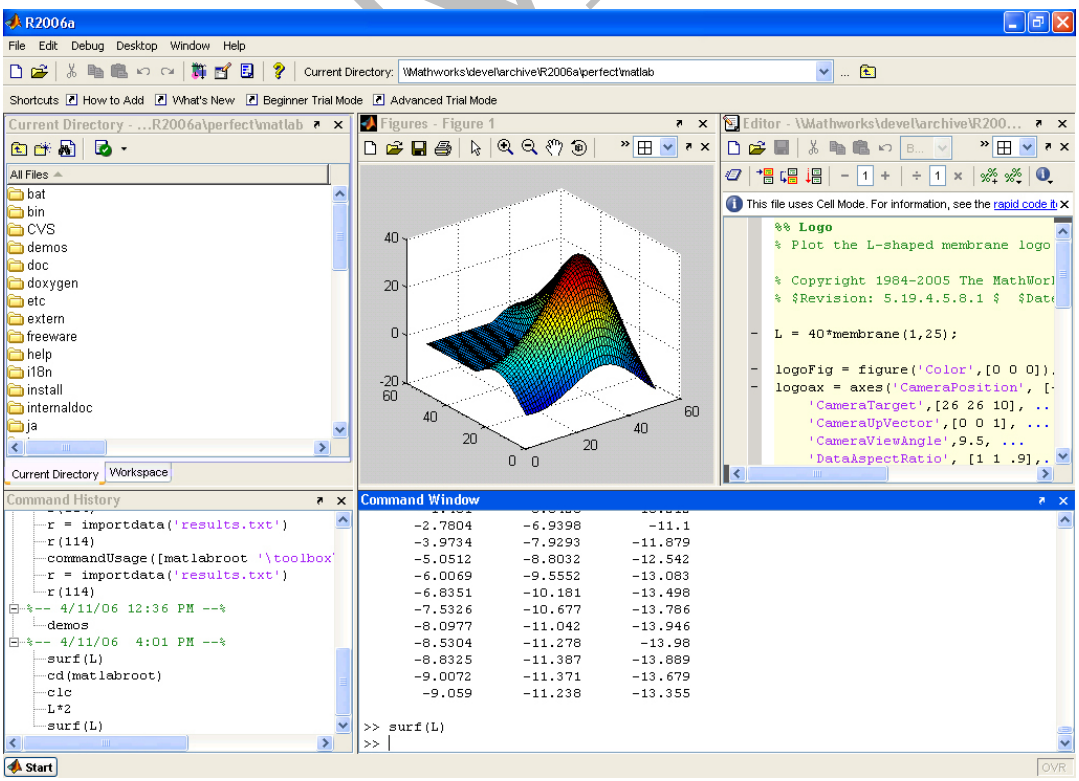
Here's a screenshot from MATLAB 5.1 in 1997:



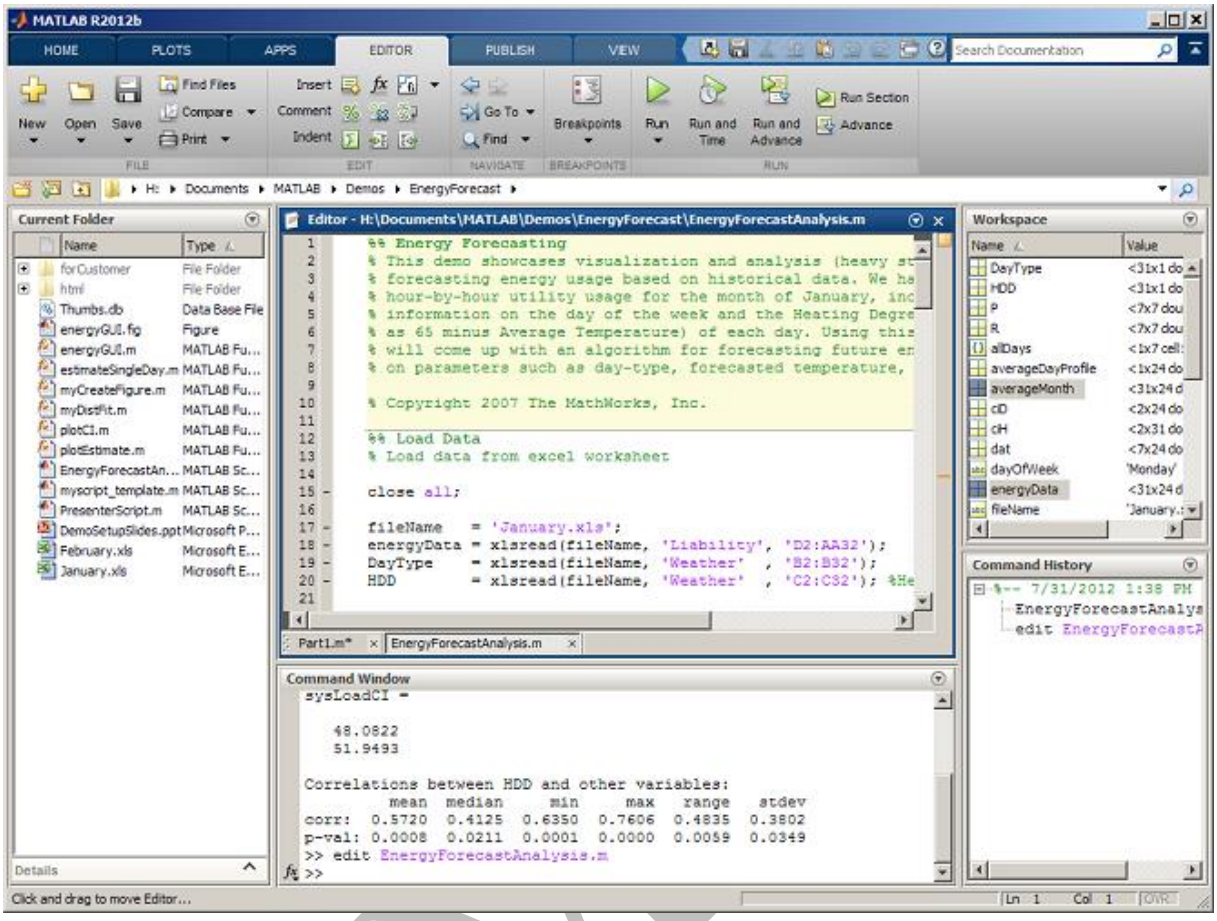
A screenshot from MATLAB 6.5 in 2002:



MATLAB 7 in 2005:



MATLAB 8 in 2012:



SAM

MATLAB core (without Toolbox)

This chapter is dedicated to all MATLAB topics that don't require any installed Toolboxes!

First thing you must know!: When you install MATLAB on a x64 O.S. then it will automatically install the x64 version of MATLAB. Same behavior if you are on a x86 system.

New main core features (to be complete)

Note that all changes are not documented here! Please refer to the official release notes for a full list.

http://www.MathWorks.ch/ch/help/matlab/release-notes-older.html

MATLAB 2014b (8.4)

- Nouveau rendu des graphiques
- Nouveau type de variables pour les dates et heures
- Support des fuseaux horaires pour les variables de dates
- Contrôle de l'angle des étiquettes des axes
- Titre des graphiques en gras par défaut
- Nouvelle syntaxe objet pour définir les propriétés des éléments de graphiques
- Modification automatique des échelles de dates lors des zoom sur les graphiques
- Création de figures à onglets avec les commandes uitabgroup et uitab (commandes anciennes mais maintenant officialisées)
- Nouvelles commandes *morebin*, *fewerbins* pour changer les bins en live sur des histogrammes
- Nouveaux categorical arrays (colonnes à facteurs) avec la commande *categorical*
- Création de Toolbox (ensemble de docs, de plusieurs apps, etc.)

MATLAB 2014a (8.3)

- Core: Pop-up Command History for recalling, viewing, filtering, and searching recently used commands in the Command Window
- Create your own icons when compiling a MATLAB app

MATLAB 2013b (8.2)

- You can now package a custom icon in an app that you create

MATLAB 2013a (8.1)

- New functions *strsplit*
- Option to add separators between controls on the quick access toolbar
- Additional icon choices, auto-scaled thumbnails, and text-formatting options for customizing descriptions of MATLAB apps

SAMPLE

MATLAB 2012b (8.0)

- Toolstrip that replaces menus and toolbars in MATLAB Desktop
- Apps gallery that presents apps from the MATLAB product family
- Suggested corrections for mistyped functions and variables in the Command Window
- Data import from delimited and fixed-width text files using Import Tool
- Single-file application packaging as a MATLAB App Installer file for inclusion in the apps gallery

MATLAB 2012a (7.14)

- New functions: *integral*, *integral2*, and *integral3*
- Transpose and Sort Variables in the Variable Editor
- *xlsread* Reads XLSX Files on All Platforms
- *VideoWriter* replace all instances of *avifile*
- *whatsnew* won't work anymore and will not be replaced

MATLAB 2011b (7.14)

- Removed function: *bessel*
- Error messages for each function now provide a link to that function's documentation.
- Under certain conditions, as you rename a variable or function, a tooltip opens. The tooltip indicates that MATLAB can rename all instances of the function or variable in the current file.
- The new Spreadsheet Import Tool allows you to select ranges of data and define rules for handling nonnumeric values

MATLAB 2011a (7.12)

- Renaming Files and Folders in the Current Folder Browser Now Reflected in the Editor

MATLAB 2010b (7.11)

- The following functions are now deprecated (but still work): *isstr*, *str2mat*, *strread*, *textread*.
- We can now customize the date format that the *Current Folder browser* and the *Command History window* use to display dates
- New functions *isrow*, *iscolumn*, and *ismatrix*
- The new *VideoWriter* function allows you to create AVI files on all platforms. As an improvement over the *avifile* function, *VideoWriter* can create files larger than 2 GB.
- For consistency with the new *VideoWriter* class, the *mmreader* class is now called *VideoReader*.
- Replace all existing instances of *wavplay* with *audioplayer* function.

MATLAB 2010a (7.10/2010)

- Create zip file from *Current Folder* panel by right click on selected files

SAMPLE

- You can now remove folders from the MATLAB path directly from the *Current Folder* panel by right click and selecting *Remove path*
- Tab completion for local variable and subfunctions names while typing scripts

MATLAB 2009b (7.9/2009)

- Start MATLAB using *-nodisplay* (UNIX) or *-noFigureWindows* (Microsoft Windows) is depreciated. Use *–nodesktop* instead.
- During a search in the Help Aa preview of where the search words were found within the page are highlighted.
- In the *Current Folder* you can use the expandable tree structure to view the contents of the current folder. Display the file type in a column.
- At the bottom of the *Current Folder* there is a small "details" that will give you some information about selected MATLAB files.
- You have now a search tool in the address bar to found a file
- MATLAB now supports PDF as the output format for publishing MATLAB code files.
- This version of MATLAB introduces a new usage for the tilde (~) operator. You can also use the tilde operator to specify unused outputs in a function call, or unused inputs in a function definition. This new syntax can help you avoid confusion in your program code and unnecessary clutter in your workspace.
- In previous releases, the Import Wizard imported only the first populated worksheet in an Excel file. The Import Wizard now allows you to specify the worksheet to import.
- New Plot Selector workspace tool that creates graphs of workspace variables

MATLAB 2009a (7.9/2009)

- Many M-Lint Messages Now Extend to Provide an Explanation and Suggested Action
- If you have installed Excel 2007 (or Excel 2003 with the Compatibility Pack) on your Windows system, the *xlswrite* function exports data to XLSX, XLSB, and XLSM formats
- Dynamic Links in Published Documents

MATLAB 2008b (7.8/2009)

- New functions: *clearvars*, *memory*
- There is a new desktop layout when you select *Desktop > Desktop Layout > Default*
- New function browser (Microsoft Excel like...) tool
- View syntax hints while entering statements
- Workspace *Browser Toolbar* is now configurable
- The new function randi returns random integers from a uniform discrete distribution

MATLAB 2008a (7.6/2008)

- On Microsoft Windows platforms, when MATLAB starts, it automatically adds the My Documents/MATLAB directory (or Documents/MATLAB on Windows Vista™) to the top of the MATLAB search path
- Customize the MATLAB desktop and possibility to customize toolbars

SAMPLE

- *Array Editor* renamed to *Variable Editor*
- The MATLAB stand-alone editor (meditor.exe) is no longer provided. Instead of the stand-alone editor, you can use the MATLAB Editor
- Specify publishing settings, such as an output directory and file format, that you can save and reuse as a group
- New *Data brushing* tool
- With the new *clearvars* function, you can specify which variables you do not want cleared from memory

MATLAB 2007b (7.5/2007)

- You can now change the MATLAB startup directory on Microsoft Windows platforms using the standard shortcut Start in field.
- New desktop layout: All but *Command Window* minimized
- Code folding feature for collapsing and expanding code
- With the *maxNumCompThreads* function, you can either set the maximum number of computational threads to a specific number, or indicate that you want the setting to be done automatically by MATLAB
- P-code files built with MATLAB 7.5 only work on 7.5 or later. They cannot be used with MATLAB 7.4 or earlier versions.
- When you save a figure, all datatips existing in it are saved along with other annotations.

MATLAB 2007a (7.4/2007)

- Introduction of MATLAB for Macintosh (Intel)
- Double-Clicking associated file type in Explorer now opens file in existing session of MATLAB Software
- Minimize, temporarily Display, and Restore panels in the desktop
- For some types of warnings or errors, M-Lint can apply an automatic fix to the code
- You can now publish an M-file function. When publishing an M-file function, you cannot evaluate the code. This feature effectively allows you to save M-file functions to output formats such as HTML or Microsoft Word documents, with formatting

MATLAB 2006b (7.3/2006)

- Introduction of MATLAB for Windows 64-bit
- The *Workspace browser* includes new columns that automatically display results of common statistical calculations: Minimum, Maximum, Range, Mean, Median, Mode, Variance, and Standard Deviation.
- Use the new *File Comparisons* tool to highlight the differences between two files
- The *beta* function, when used with 3 inputs won't work anymore
- The *gamma* function, when used with 2 inputs won't work anymore
- When using *whos* there is a new column in the output called *Attributes* that identifies values that are sparse, complex, global, or persistent.
- When using *whos* the words "array" and "object" have been dropped from items under the Class heading. Items formerly listed as double array or timer object, for example, are now displayed as double, and timer.

SAMPLE

- When using *whos* MATLAB no longer includes summary information showing the "Grand total" of elements and bytes at the bottom of the *whos* output display.
- There is an additional field in the structure returned by *whos*. This new field is *'persistent'* and is set to logical 1 for those variables that are persistent, or logical 0 otherwise.
- Using the new *first* option with the *unique* function returns a vector with elements that represent the lowest indices of unique elements in the input vector.
- Save *Compression* and *Unicode* Options Removed. Instead, save now creates a MAT-file with compression and Unicode character encoding by default.
- The function definition line in a function M-file no longer requires commas separating output variables.
- Printing MATLAB figures has become easier as a result of combining the *Page Setup*, *Print Setup*, and *Print Preview* dialogs into one tabbed *Print Preview* dialog.
- If your mouse has a center scroll wheel, you can use it to zoom in and out of axes on charts, as well as by clicking and/or dragging
- Data Cursor text can now be programmatically modified

MATLAB R14SP3 (7.1/2005)

- The behavior of MATLAB software when started on Microsoft Windows platforms with the *-nodesktop* option has changed. The MATLAB Command Window no longer displays a menu bar or toolbar
- The Editor/Debugger now supports a horizontal or vertical split screen for displaying two different parts of the same document at once.
- You can set a preference to highlight the current line, that is, the line with the caret
- New function *hypot* for the square root of sum of squares
- New function *mod* to find the most frequent values in sample
- The *rand* function supports Mersenne-Twister algorithm
- The *datestr*, *datenum*, and *datevec* functions now support time specification in milliseconds

MATLAB R14SP2 (7.0.4/2005)

- The Installer honors the default installation folder for Windows software, which on most machines is *Program Files*.
- Go directly to a subfunction or nested function within an M-file using the enhanced *Go To* dialog box
- In MATLAB versions prior to R14, date values read into MATLAB from an Excel spreadsheet using xlsread were always imported as numeric date values. The R14 and later releases of MATLAB import dates in the format in which they were stored in the Excel file. Dates stored in string or date format are now imported as strings by xlsread. Dates stored in numeric format are imported as numeric date values.
- You can display MATLAB output using two new formats: *short eng* and *long eng*.
- The *imwrite* function now supports exporting image data in Graphics Interchange Format (GIF)

SAMPLE

MATLAB R14SP1 (7.0.1/2004)

- Nothing new interesting in my point of view...

MATLAB R14 (7.0/2004)

- A 64-bit version of MATLAB is now available for Linux platforms based on AMD64 and Intel EM64T processors. A 64-bit processor provides a very large amount of available memory, up to 2^{64} bytes = 18,446,744,073,709,552, 000 bytes (16 exabytes), enabling you to store a very large amount of information.
- The largest contiguous block of memory under Windows XP to approximately 1.5 GB, equivalent to 180 million double precision values
- New functions: *addtodate*, *intmax*, *intmin*, *isfloat*, *isinteger*, *isscalar*, , *isvector*, *recycle*, *restoredefaultpath*, *strtrim*, *textscan*, *xlswrite*, *isstrprop*
- New *feature('memstats')* to see avalaible memory on the system
- Now MATLAB is case-sensitivity in functios and directory names for MS Windows and UNIX (was only case sensitive on UNIX before).
- Anonymous functions give you a quick means of creating simple functions without having to create M-files each time. You can construct an anonymous function either at the MATLAB command line or from within another function or script.
- When converting between serial date numbers, date vectors, and date strings with the *datenum*, *datevec*, and *datestr* functions, you can specify a format for the date string
- You can now specify the range to import with *xlsread*
- MATLAB now includes a function, named *mmfileinfo*, that returns information about the contents of a multimedia file. The file can contain audio data, video data, or both
- From within MATLAB, you can connect to an FTP server to perform remote file operations.
- MATLAB can now consume Simple Object Access Protocol-based (SOAP) Web services with the createClassFromWSDL function.

MATLAB R13 (6.5/2000)

- MathWorks products in Release 13 no longer support Microsoft Windows 95.
- This release introduces the Profiler, a new graphical user interface for assessing the performance of your M-files so you can make performance improvements
- Display of line number, column number, and current function in the Editor
- Colormap editor enables interactive colormap editing
- New text properties provide control over the text background
- Redesigned property editor accesses graphics properties with interactive editor

MATLAB R10 (5.2/1998)

- New MATLAB language functions, implementing features such as try/catch error handling
- Support for two new ActiveX technologies: ActiveX control containment and ActiveX Automation client capabilities, so that now MATLAB can both control and be controlled by other ActiveX components

SAMPLE

- Support for OpenGL rendering to improve performance dramatically for many visualization application
- Enhanced visualization features, including enhanced camera control, simplified placement of Light objects, tighter and more consistent control of graphics object hiding.

MATLAB R9 (5.1/1997)

- New scatter plot functions

MATLAB R8 (5.0/1996)

- Can't use logical (binary) vector anymore to subselect a vector or matrix
- Integrated M-file editor
- Visual M-file debugger
- M-file performance profiler
- Search path browser/editor
- Workspace browser
- Web-based online Help Desk/documentation viewer
- GUI builder
- New multidimensional arrays
- New User-definable data structures
- New cell arrays: multitype data arrays
- New Character arrays: two bytes per character
- Switch/Case statements
- Import/Export images functionality
- Greeks symbols available in charts
- Dual axis plots
- Three dimensional quiver, ribbon and stem plots
- Pie charts and 3D bar charts
- Extended curve make symbol family

SAMPLE

Cite MATLAB

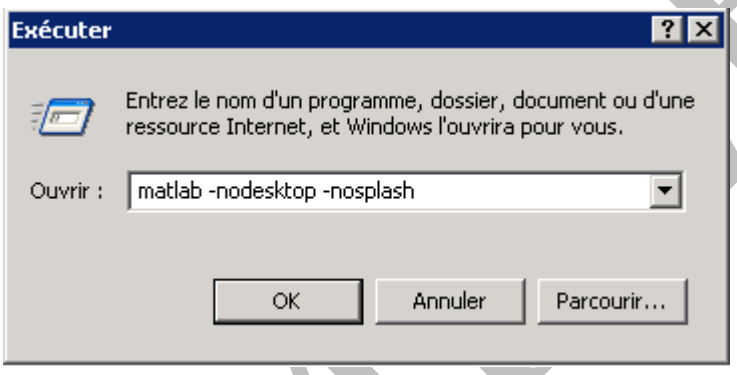
To cite MATLAB (and in this case a toolbox) you can use this:

MATLAB and Statistics Toolbox Release 2012b, The MathWorks, Inc., Natick, Massachusetts, United States.

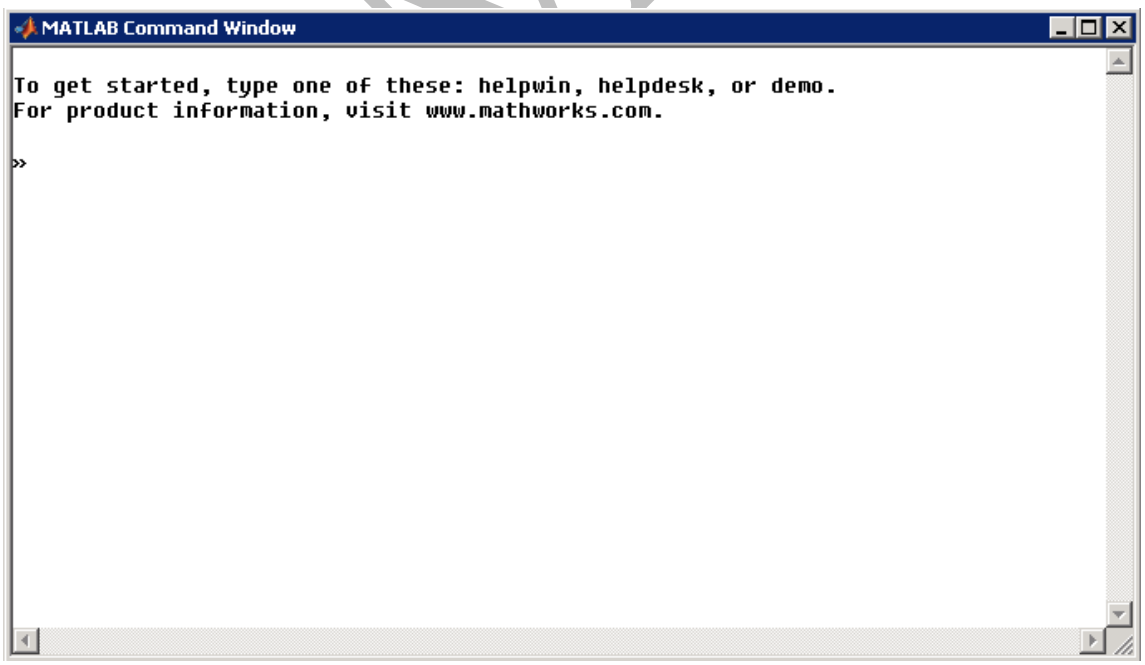
User Interface

The goal of this chapter is to learn the basic of MATLAB without using any Toolboxes!

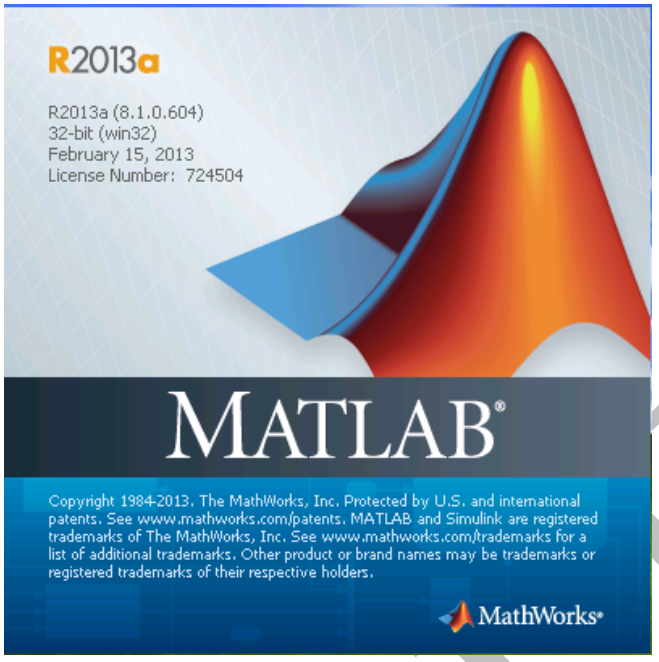
To open MATLAB without the GUI in the MS Windows command prompt type the following:



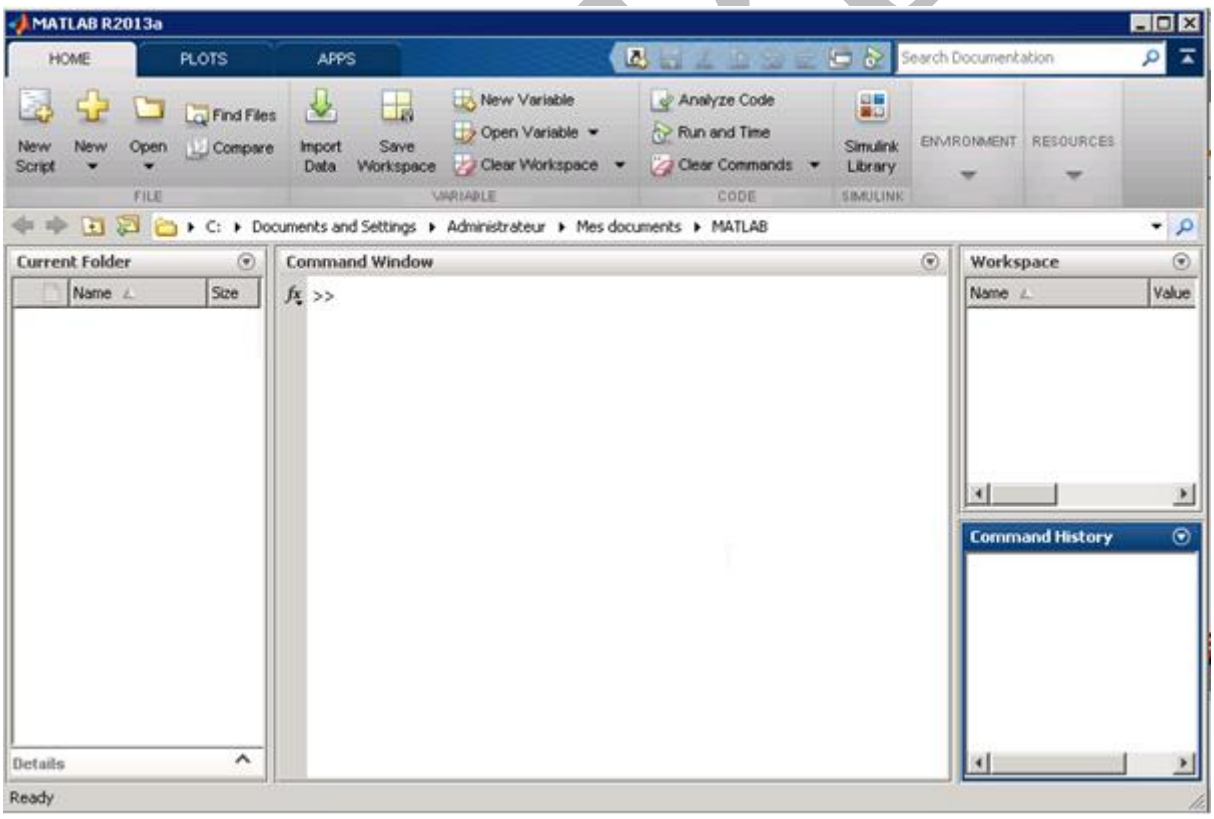
You will get the following old style robust scientific (UNIX like) interface:



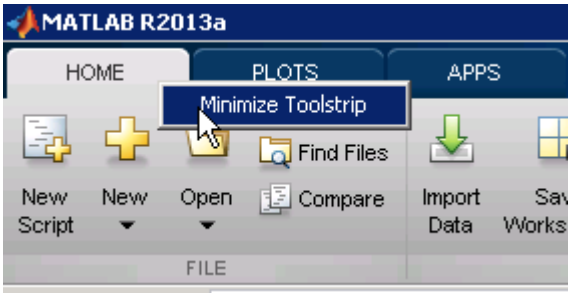
If you run MATLAB by the standard way you will first get the splash screen during a few seconds (here MATLAB 2013a splash screen):



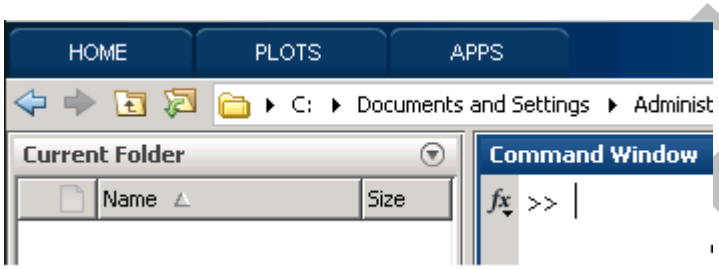
And after you will see the MATLAB GUI open:



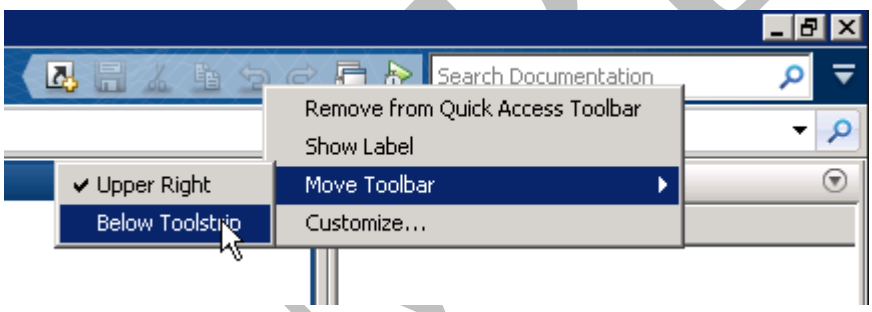
You can minimize the ribbons like in Microsoft Office by double clicking on the registrer or by makin a right click:



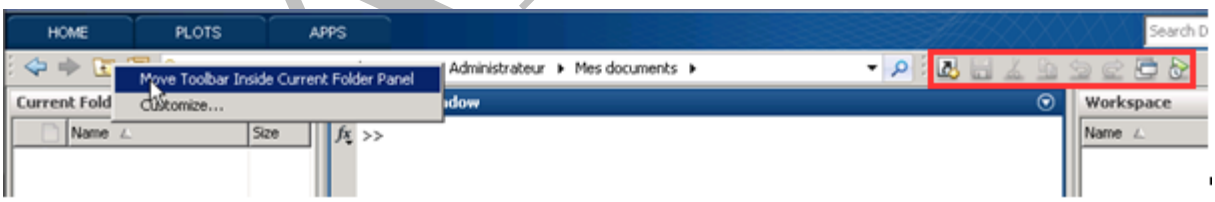
The result will be:



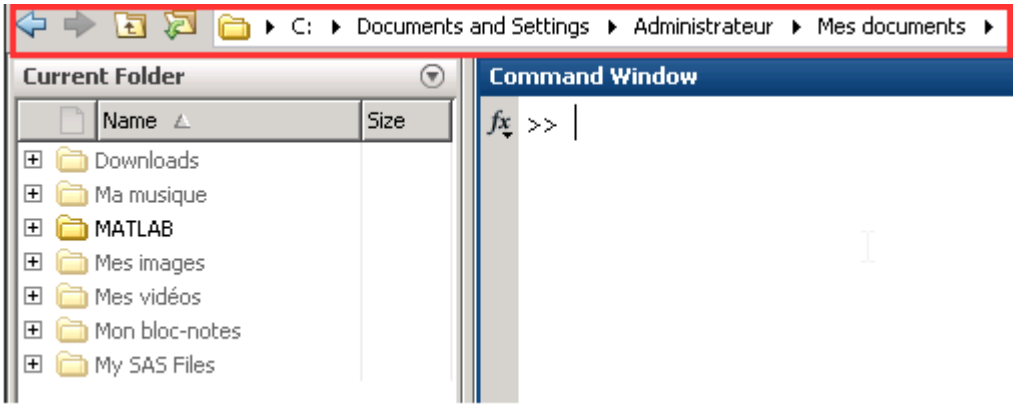
You can also change the position of the **Quick Access Toolbar**:



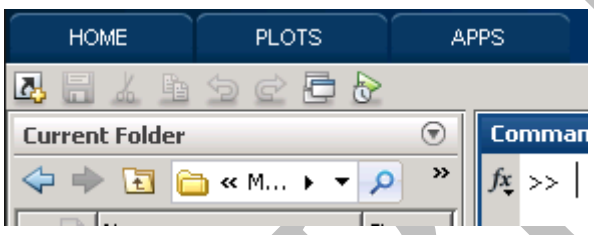
Or take it back to its previous position:



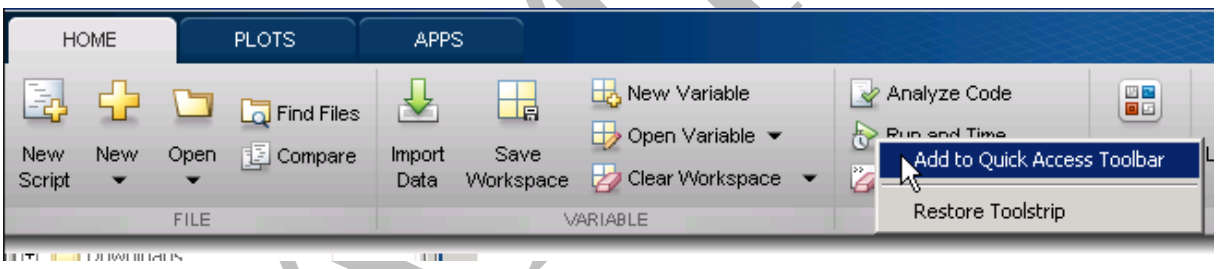
As you can see MATLAB has a **Path browser toolbar**:



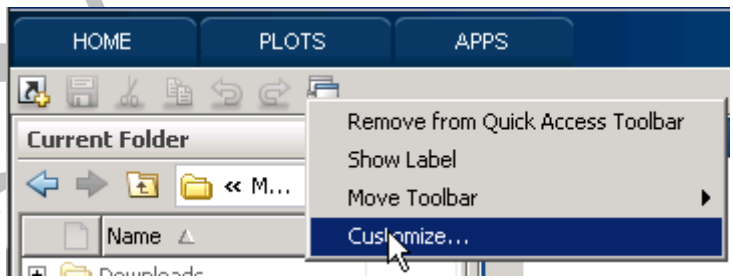
With a right click you can move it into the **Current Folder Panel**:



Like in Microsoft Office you can move you favorites buttons to the Quick Access Toolbar by right click and selecting **Add to Quick Access Toolbar**:

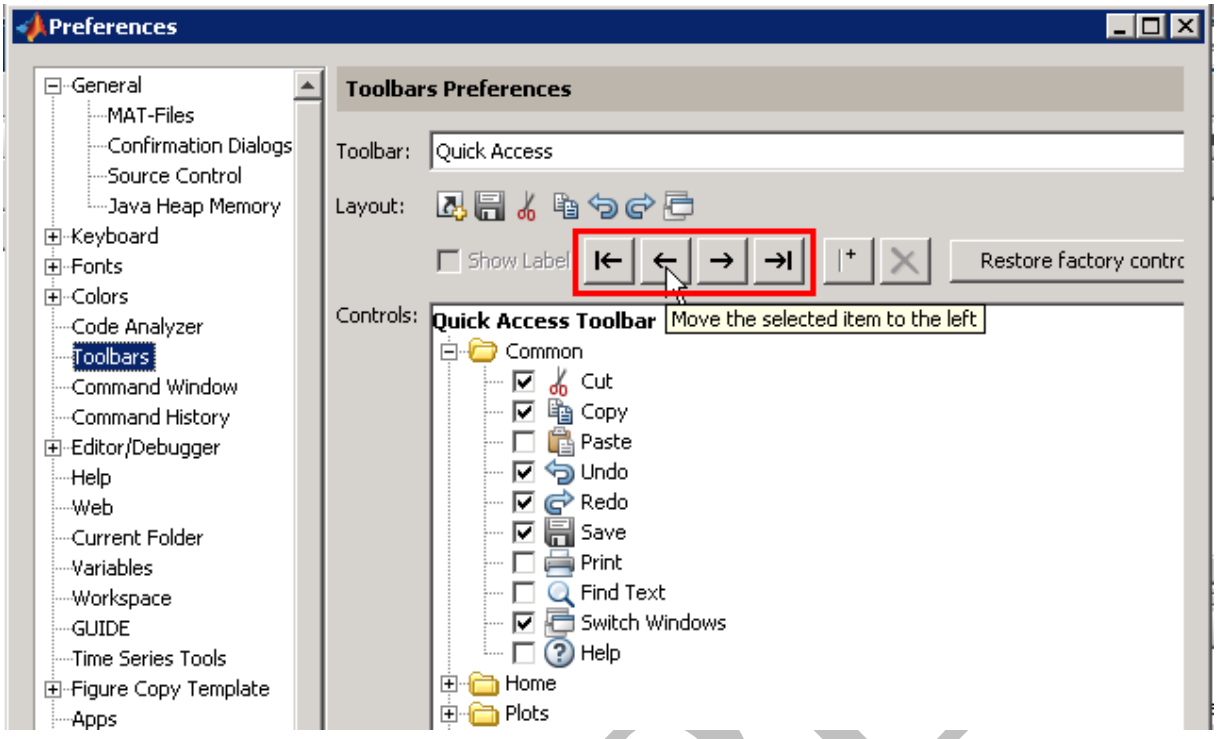


You can also like in Microsoft Office customize the Quick Access Toolbar:

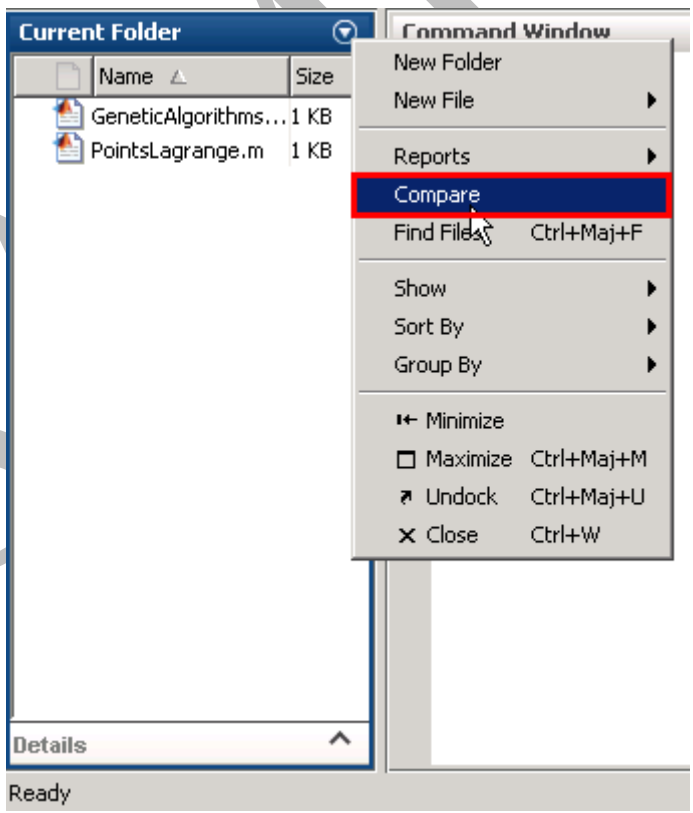


And you will get a windows that looks very similar to the Microsoft Office one:

SAMPLE



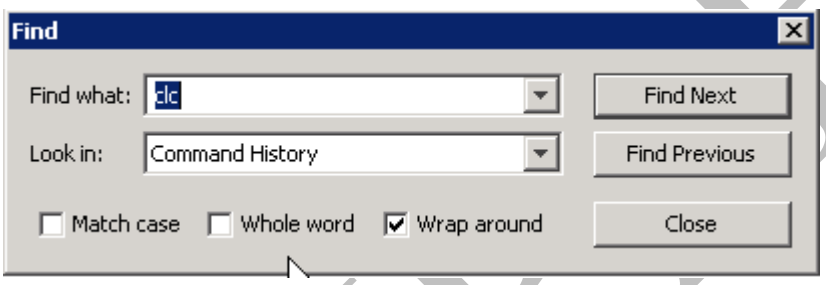
Through the **Current Folder panel** you can if you wish to compare two script files when you click on the small arrow on the top right corner:



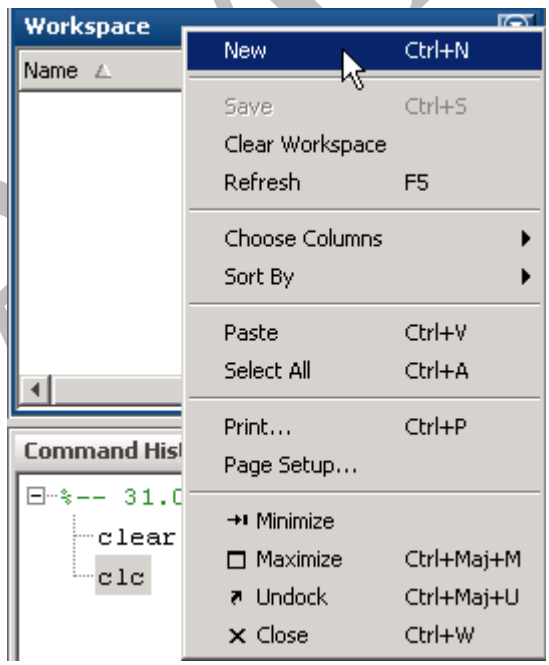
If you click on the small arrow of the top right corner of the **Command Window** you can **Find...** (search) previous type commands. This is very useful!:



You will get the following famous dialog box on you screen where you can type some keywords to search a previous typed command:

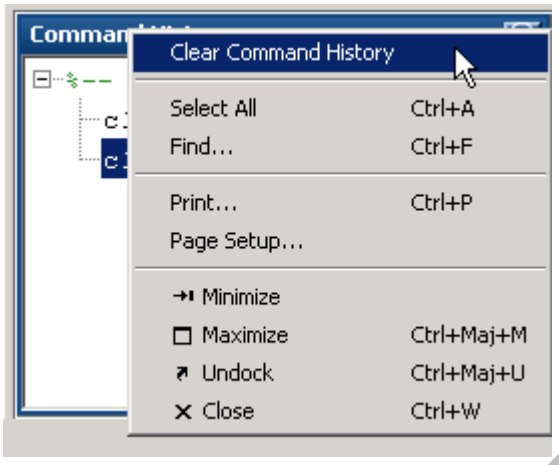


Through the **Workspace Panel** small icon you can also create new script files (most people use instead **Ctrl+N** for this purpose):

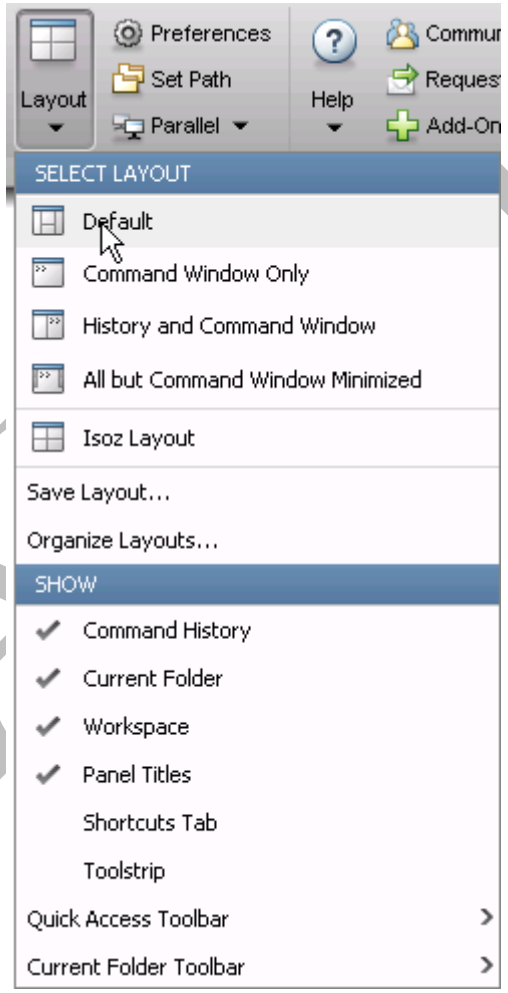


In the **Command History Panel** you can clear the history if you wish to:

SAMPLE

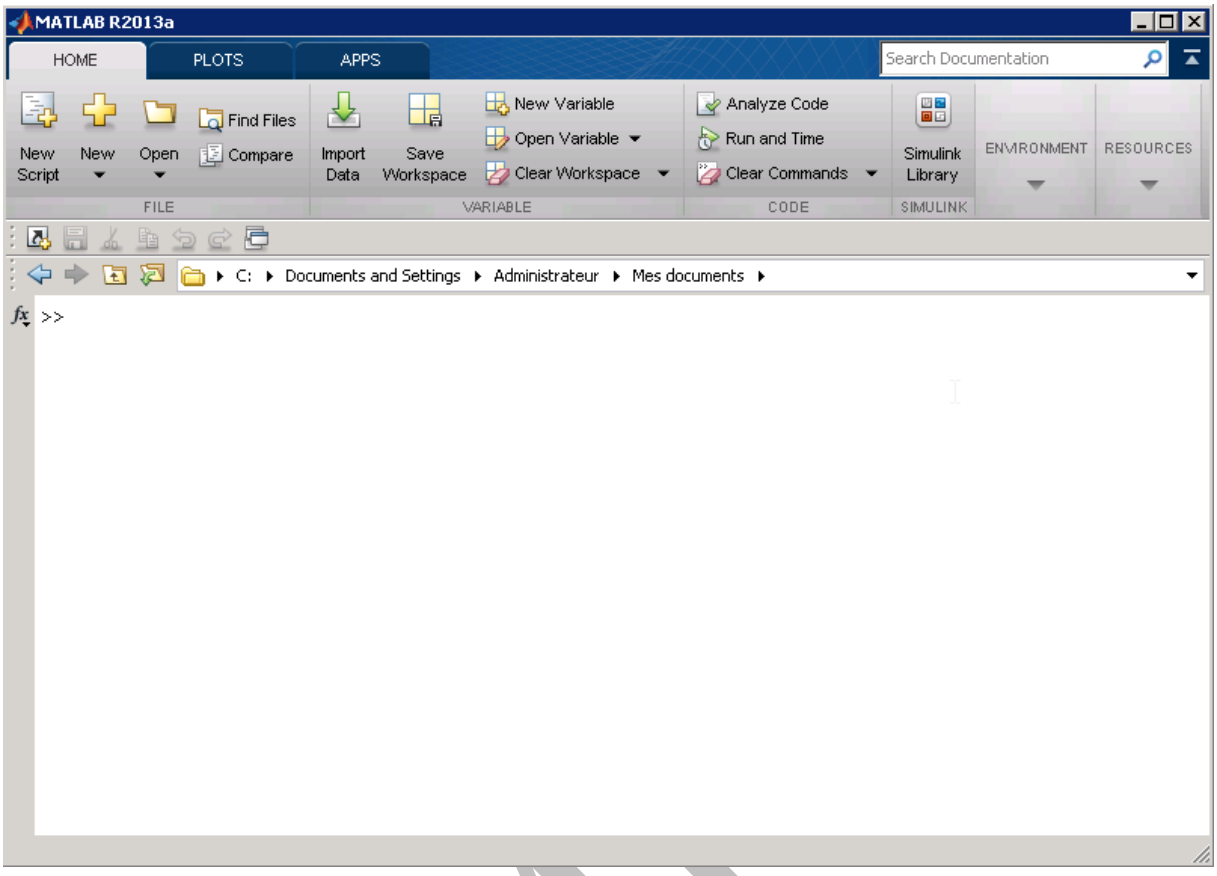


You can also choose between different presentation layout of the GUI:

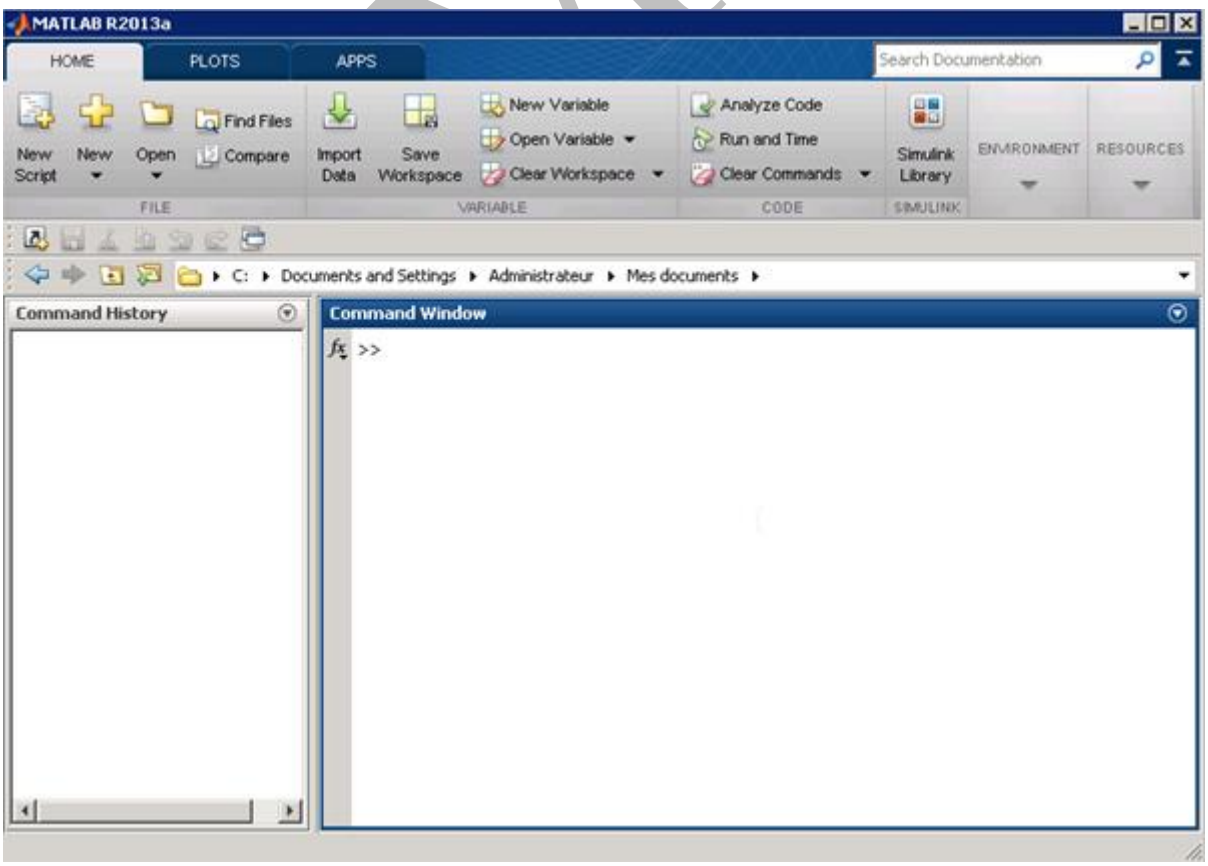


Here below you have for example the **Command Window Only** layout visible:

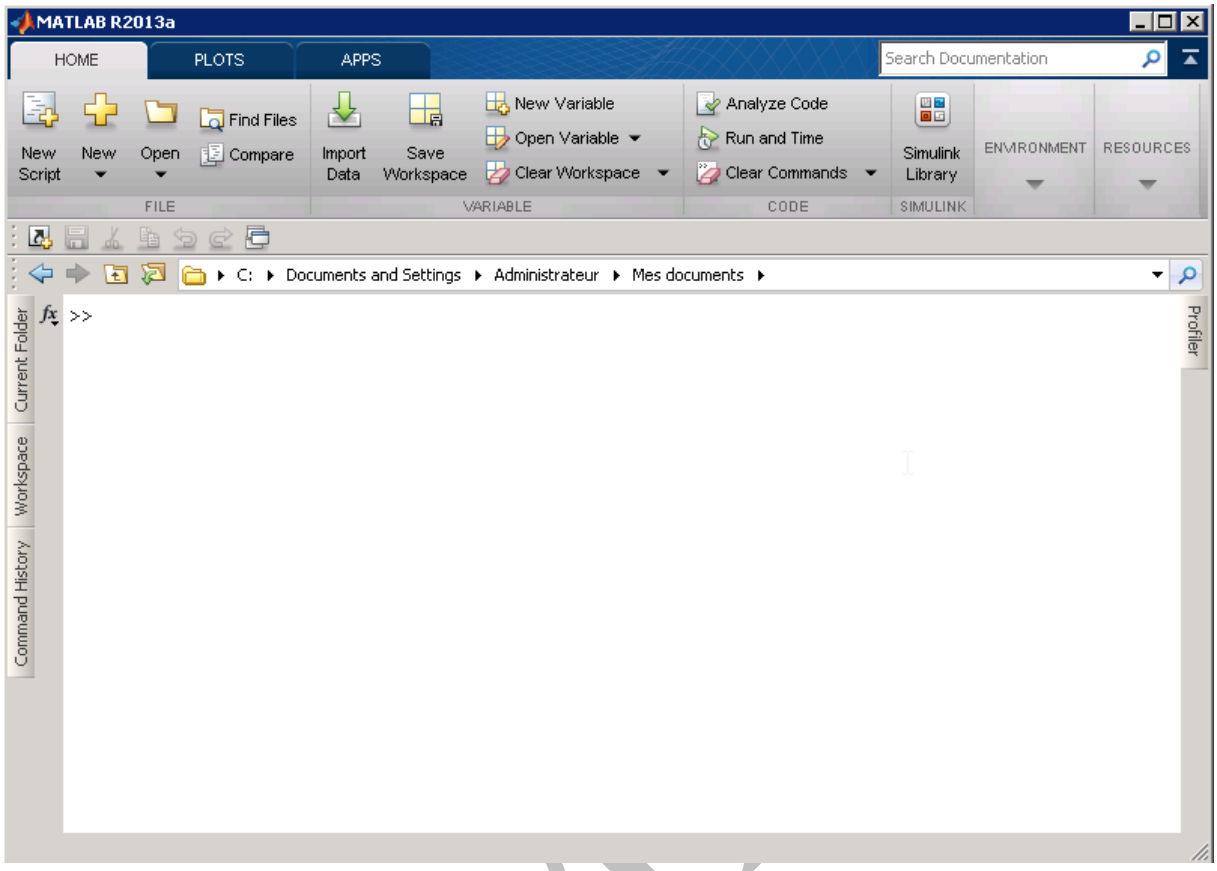
SAMPLE



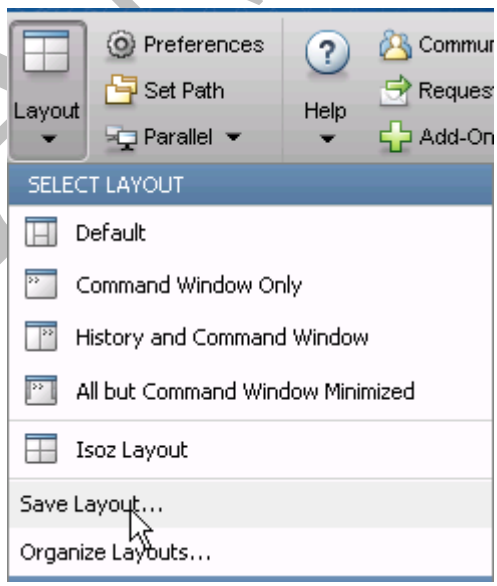
And here the **History and Command Window** layout:



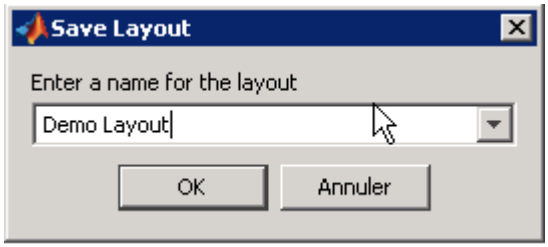
Or the **All but Command Window minimized**:



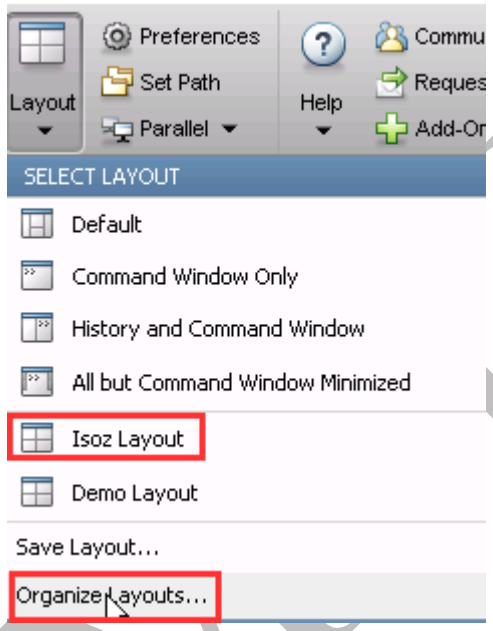
You can also save you customized layout because you can drag & drop the panels with the mouse. For this click on **Save Layout...**:



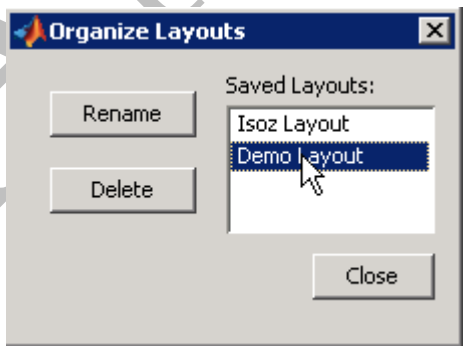
The software will ask you to give a name for the layout:



Once saved you will see your layout in the **Layout** list:

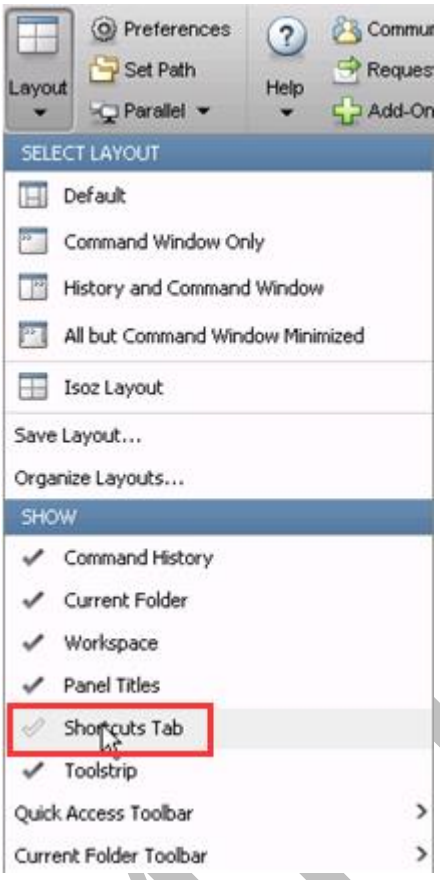


If you click on **Organize Layouts...** you will have the possibility to rename or remove one of the Layout you have created:

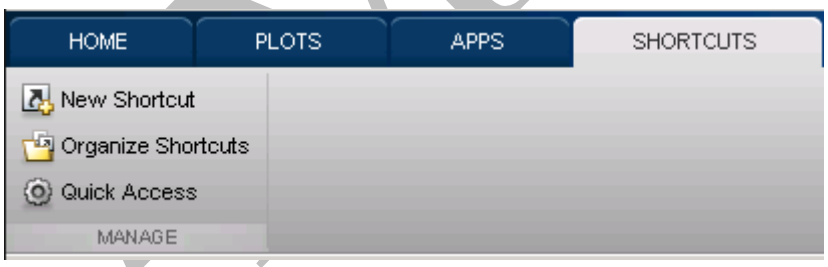


MATLAB has a hidden tab (register) that allows you to create your own shortcuts to activate it click on **Shortcut Tab** in the **Layout** button:

SAMPLE

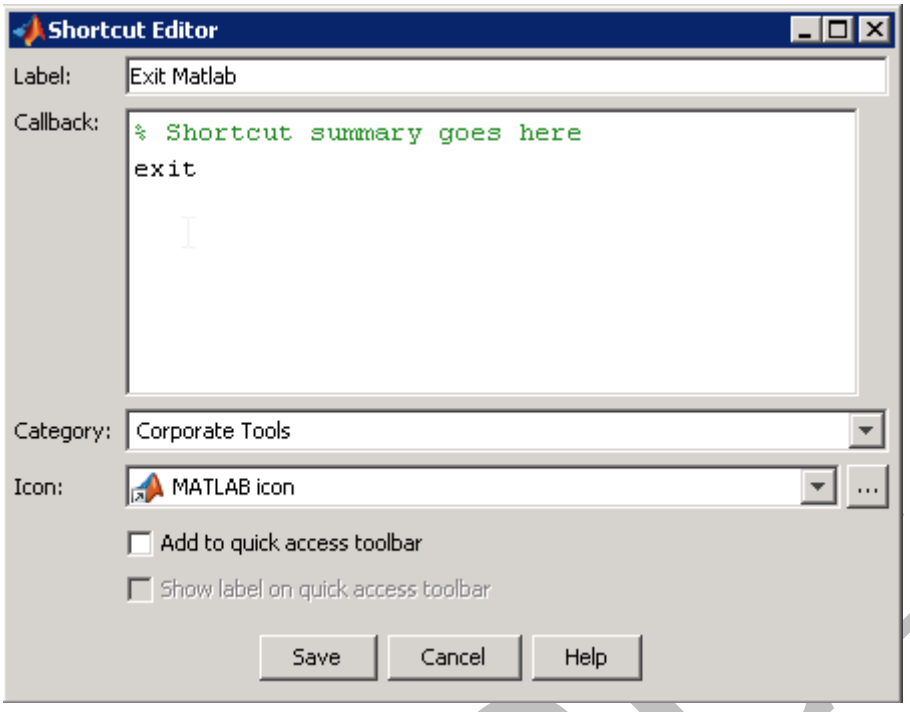


Then you will see a new tab as following:

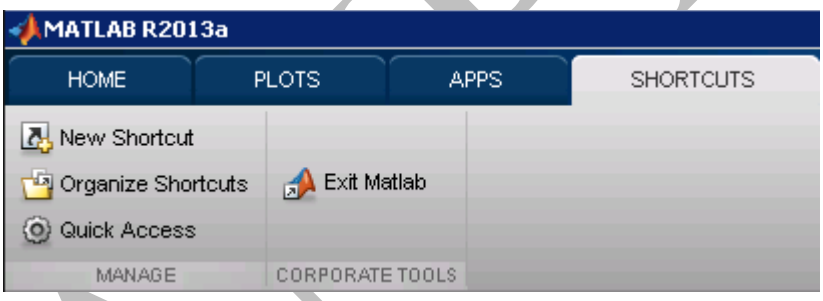


To create a new shortcut just click on **New Shortcut**...:

SAMPLE



and type the informations you want and click **Save**. Then you will get:

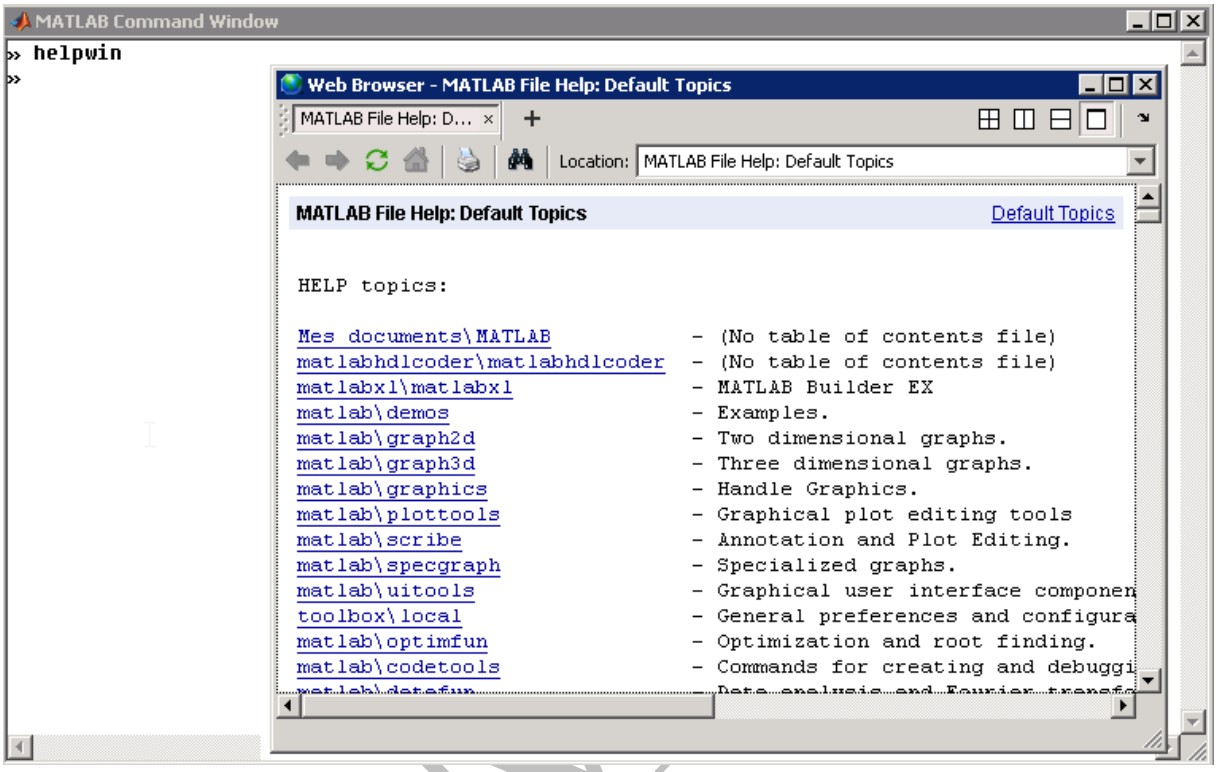


SAMPLE

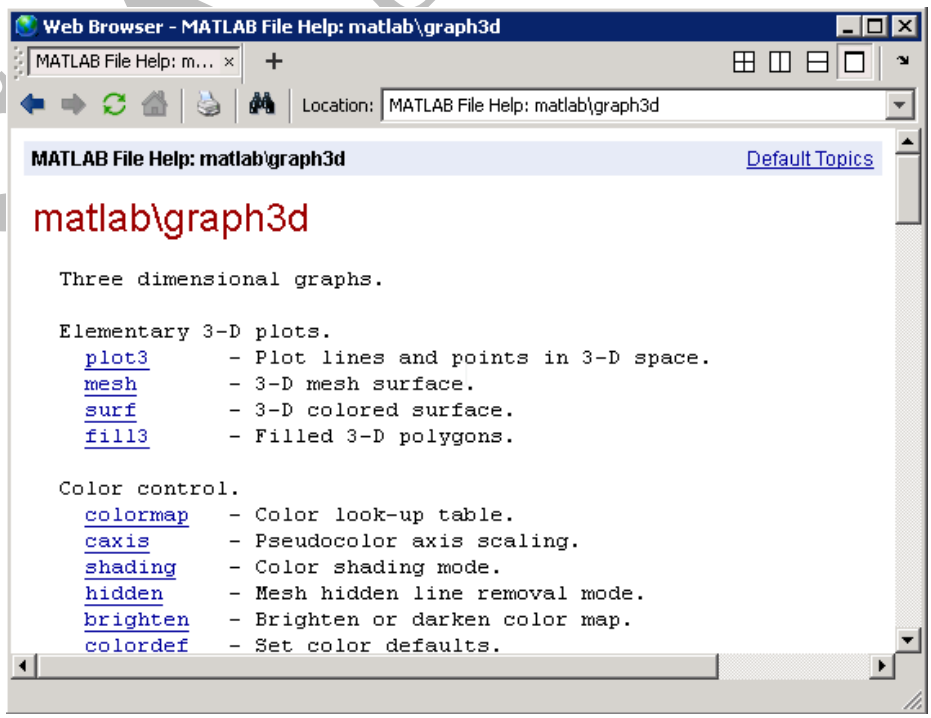
Use Help

Using the help in MATLAB is very important because there is more than 8'000 functions and you can't remember and now everything.

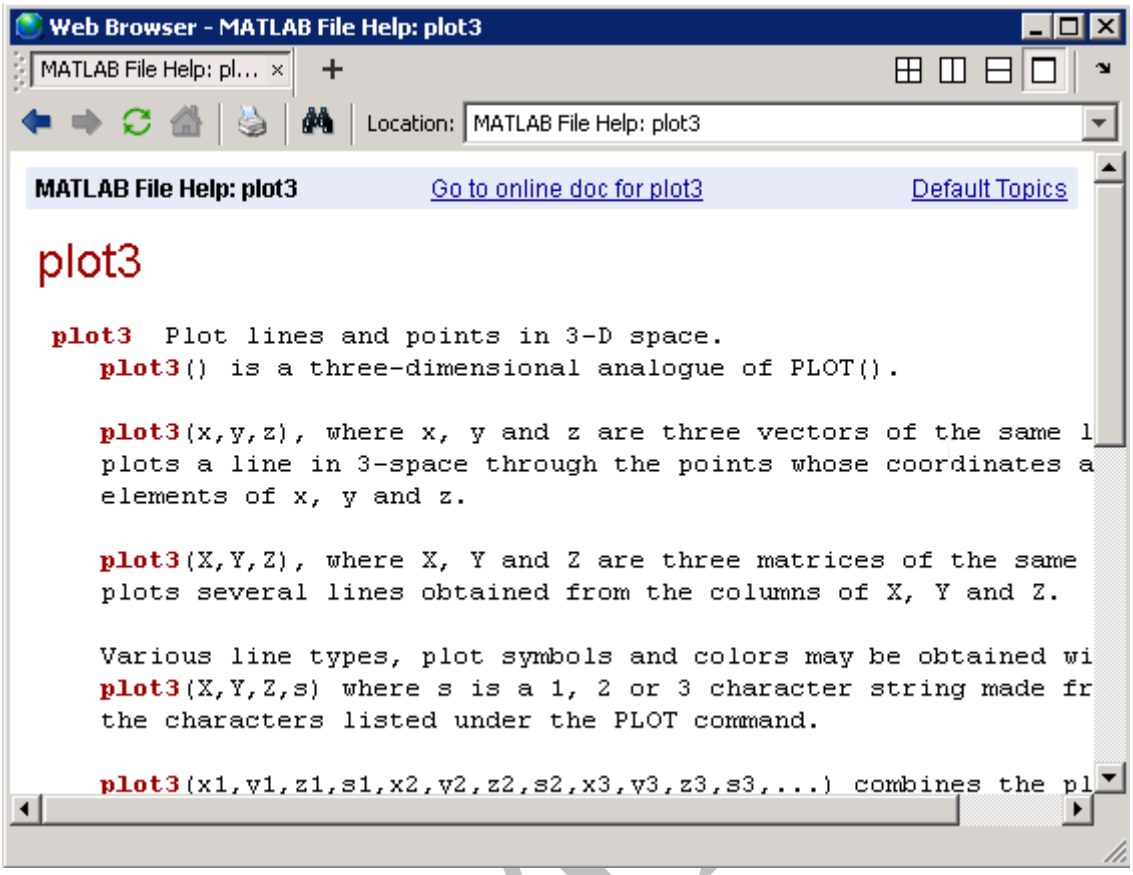
To open the classic help window use the command **helpwin**:



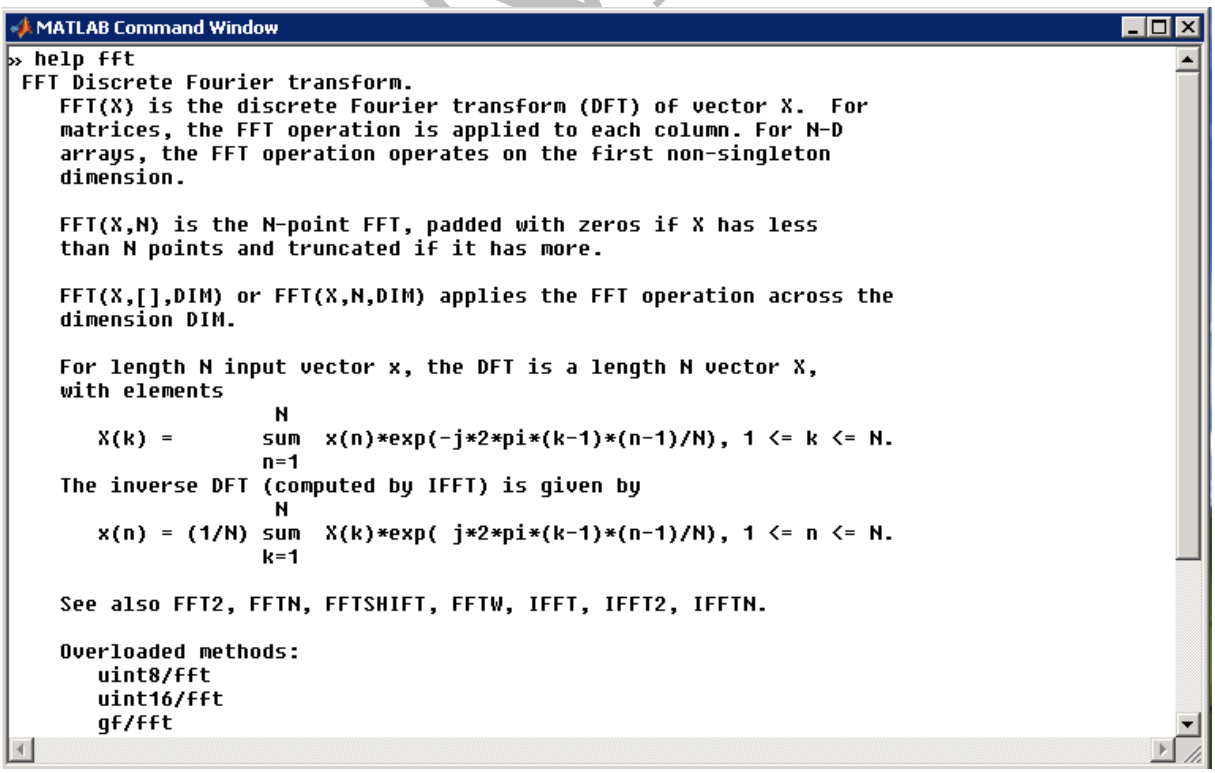
And if you click on one of the subjects you will get something like this:



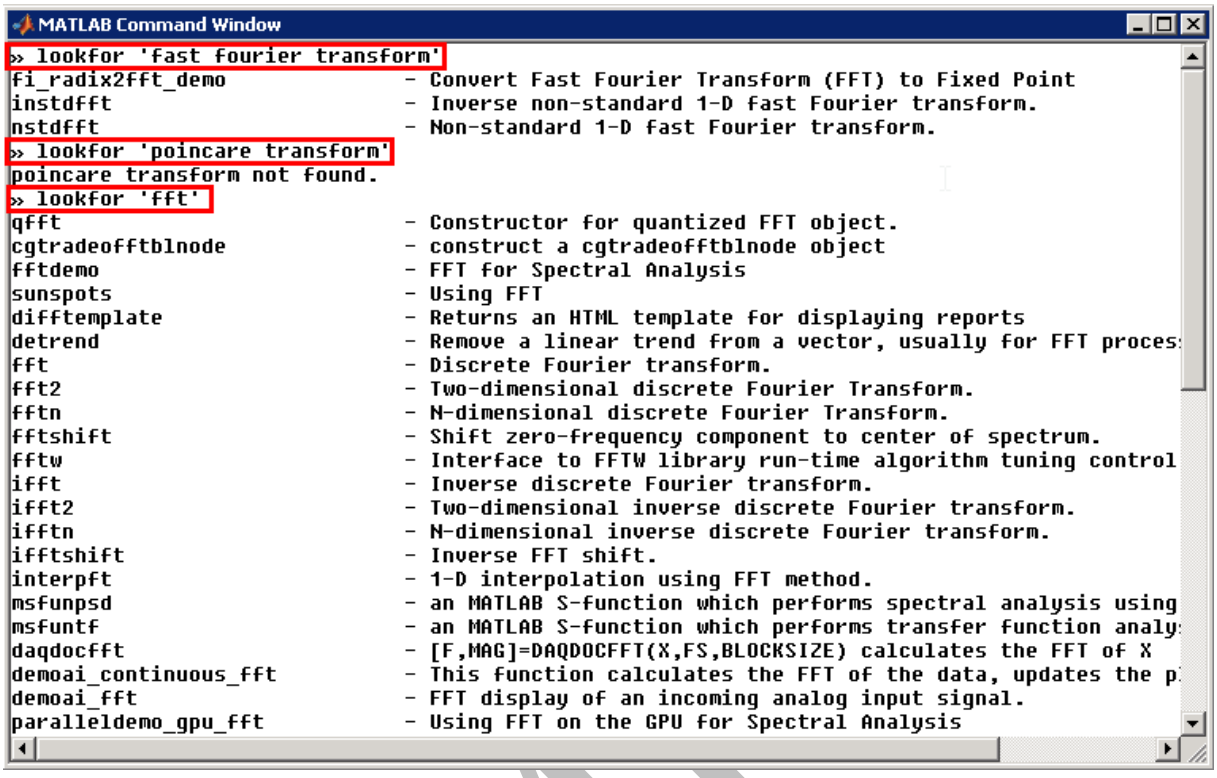
And if you click once again on one of the sub-subjects you will get something like this:



In reality the best way to get a quick answer and help is to use the command **help**:

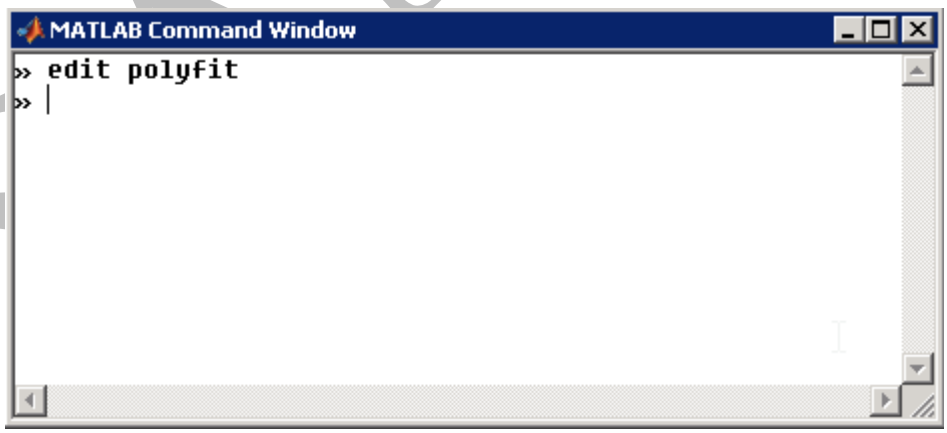


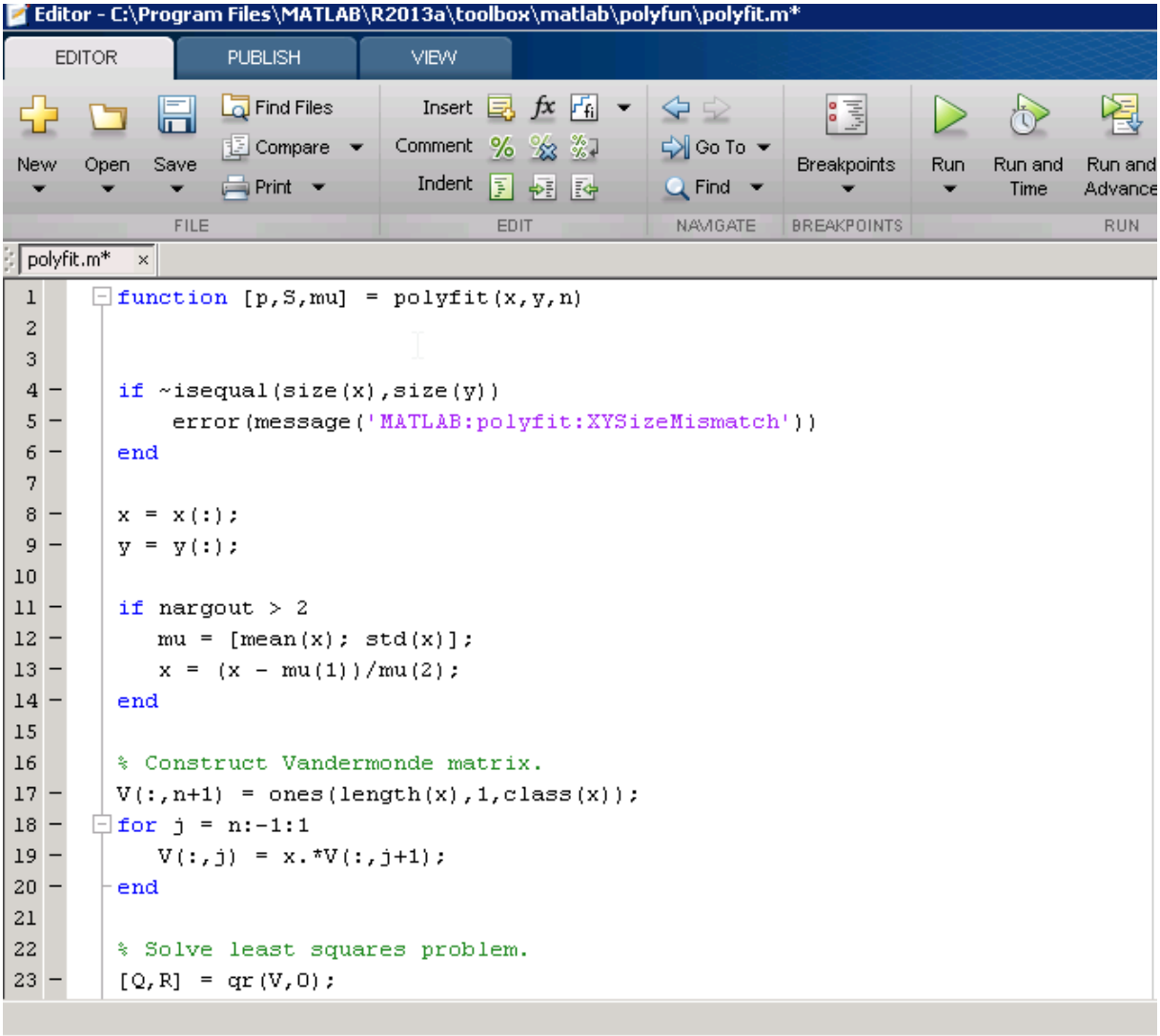
And if you don't know the name of what you are looking for you can use the commande **lookfor**:



Edit a native function source code (if available)

The source code of some MATLAB functions are available. In some cases this is especially useful so you can made modifications to perform or adapt the existing script to your needs (practice to use only in case of emergency solution!):




Editor - C:\Program Files\MATLAB\R2013a\toolbox\matlab\polyfun\polyfit.m*
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Insert
Comment
Indent
EDIT
Go To
Find
NAVIGATE
Breakpoints
BREAKPOINTS
Run
Run and Time
Run and Advance
RUN
polyfit.m*
1 function [p,S,mu] = polyfit(x,y,n)
2
3
4 if ~isequal(size(x),size(y))
5 error(message('MATLAB:polyfit:XYSizeMismatch'))
6 end
7
8 x = x(:);
9 y = y(:);
10
11 if nargout > 2
12 mu = [mean(x); std(x)];
13 x = (x - mu(1))/mu(2);
14 end
15
16 % Construct Vandermonde matrix.
17 V(:,n+1) = ones(length(x),1,class(x));
18 for j = n:-1:1
19 V(:,j) = x.*V(:,j+1);
20 end
21
22 % Solve least squares problem.
23 [Q,R] = qr(V,0);

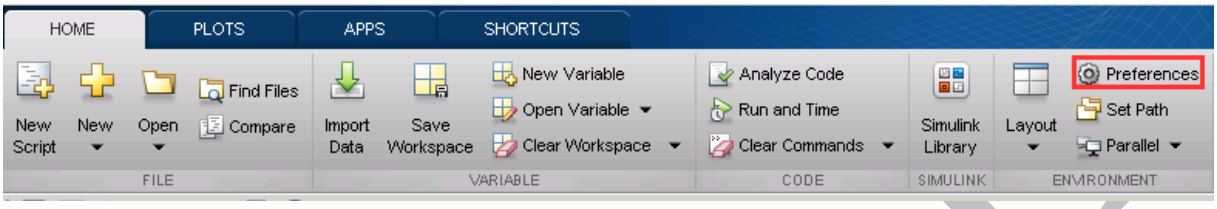
SAM

Most asked user preferences

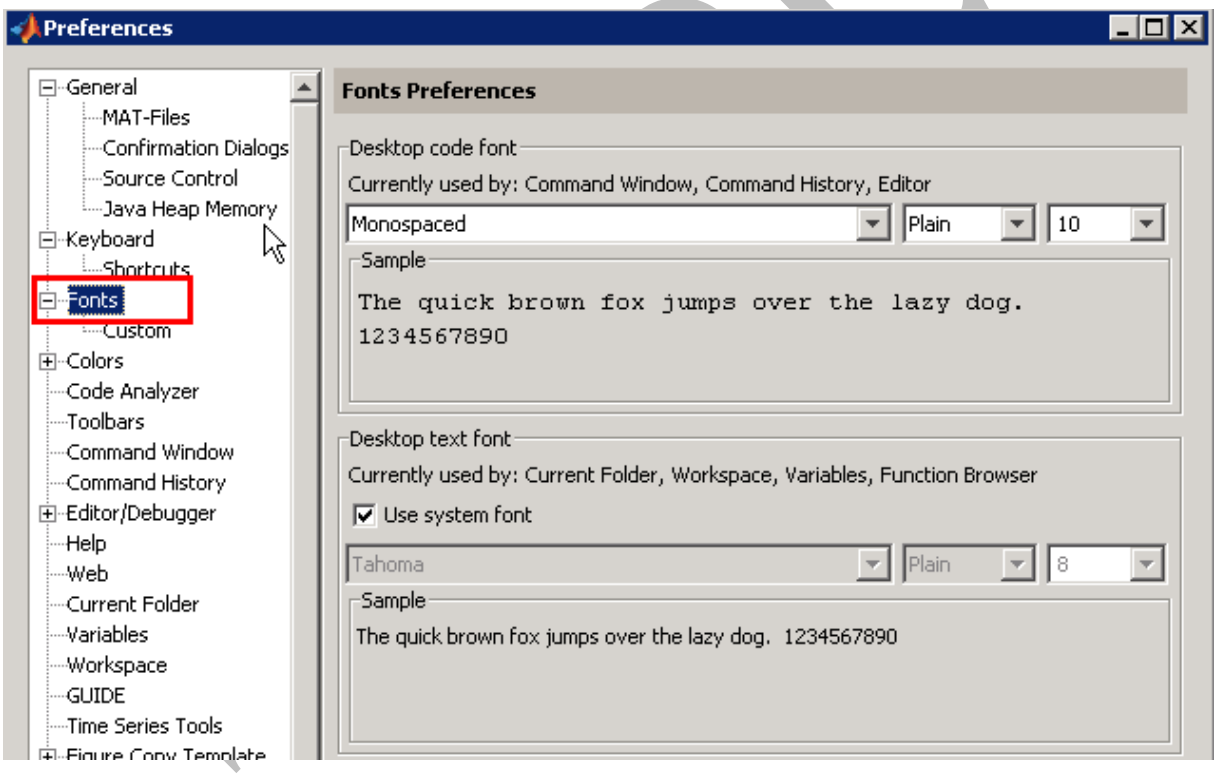
Here we would like to show you only a few MATLAB preferences (system settings) asked by large number of practitioners.

The first setting is how to change the font in the Command Window, Command History and Editor.

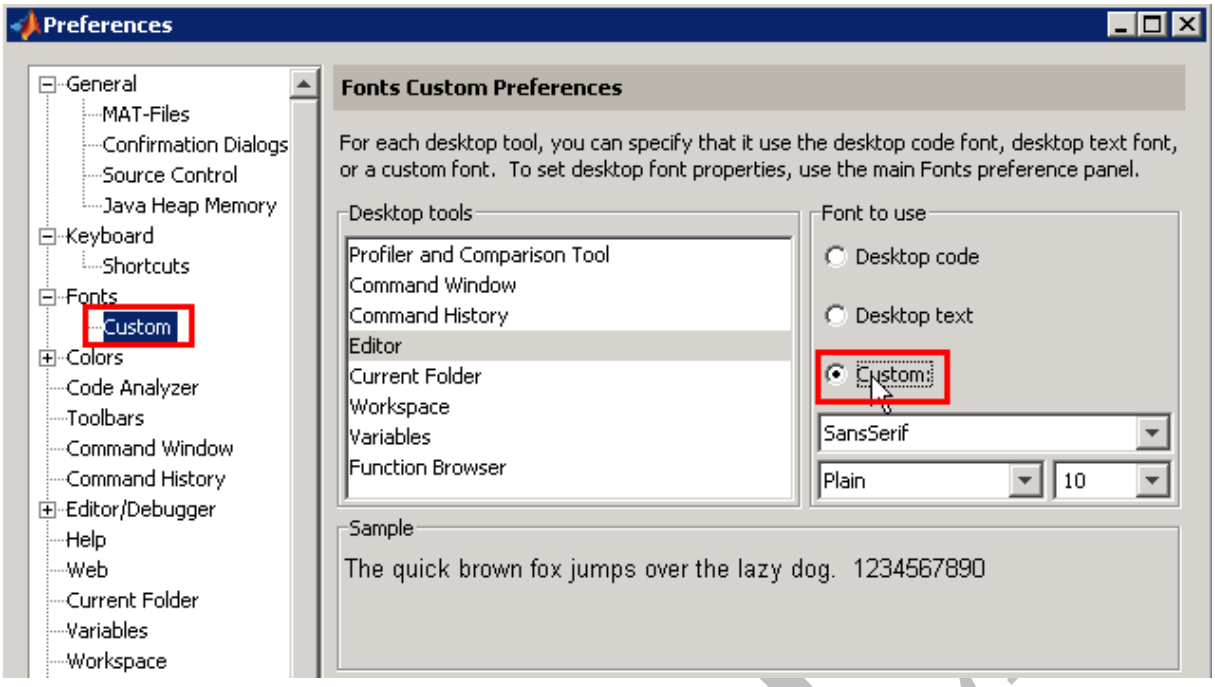
First you go in the tab **Home** and you click on **Preferences**:



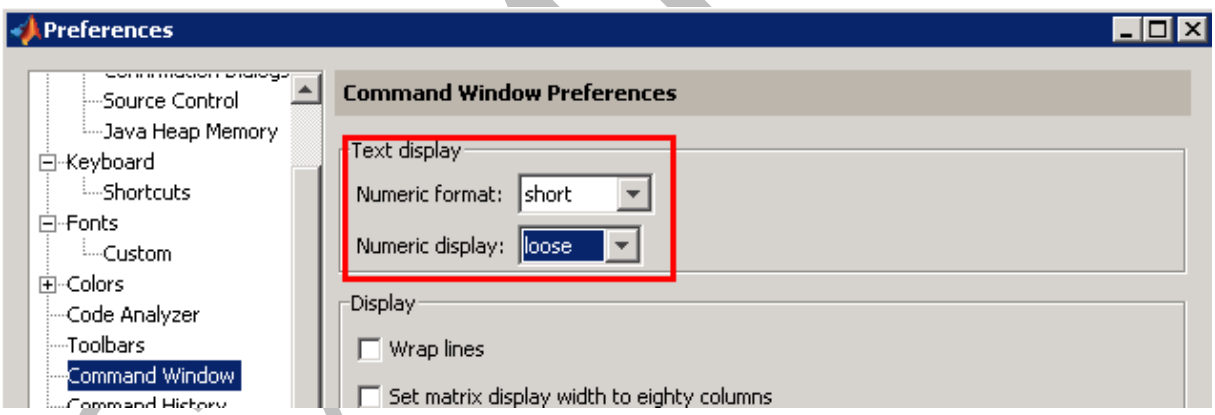
And when the properties windows opens select the **Fonts** section and change what you want the way you want:



If you want to select a special font apart for the Editor you have to click on the **Custom** section inside of **Fonts** as you can see below:



Another annoying point in MATLAB editor for a lot of people is the default display of the **Command Window**. By default format are short and the numeric display is loose but you can change this to more accurate and optimal choices:

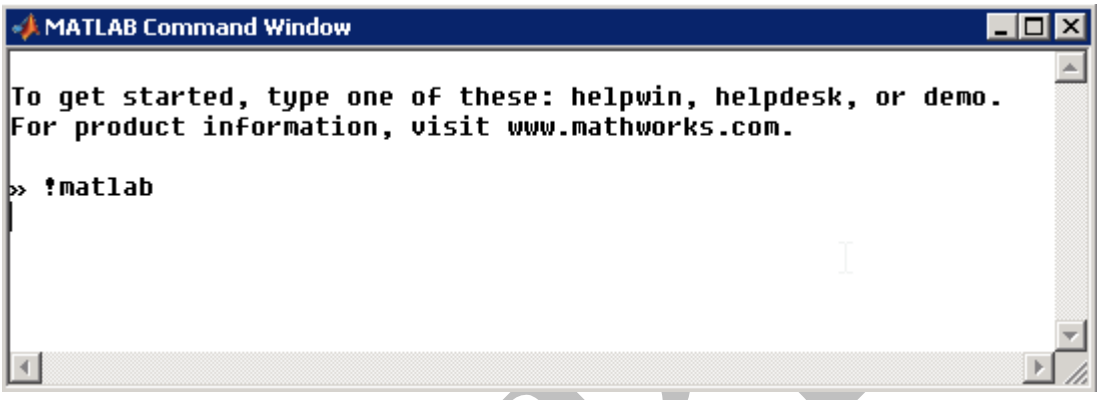


General commands

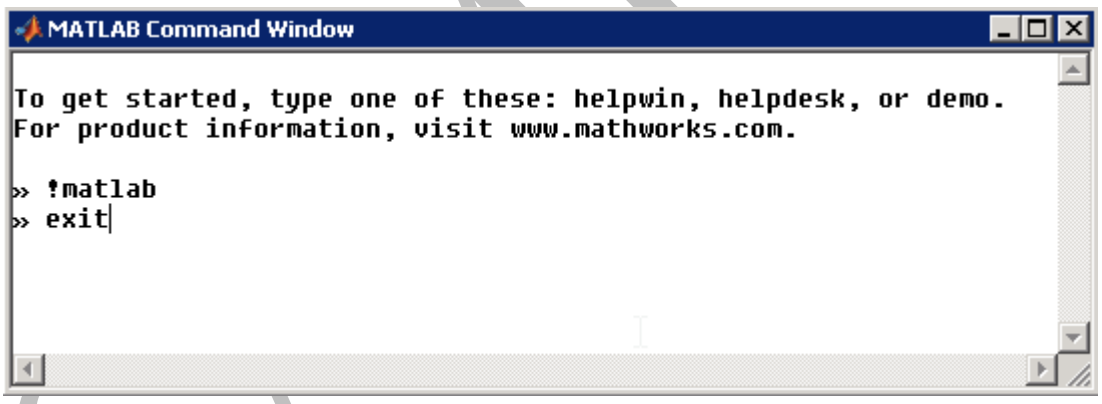
Here we will just take a look to some very general MATLAB commands that are a **must know**!!

Execute system commands

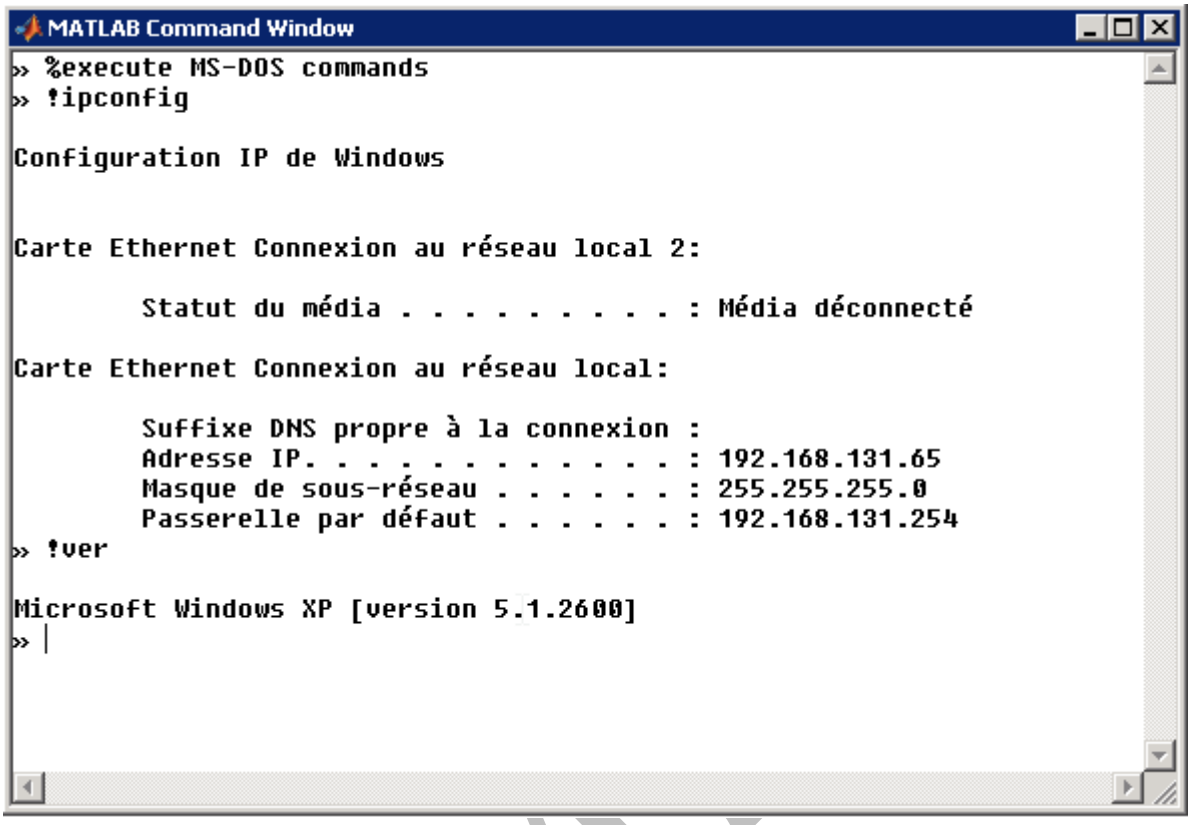
First if you write **!MATLAB** from the old style Command Window you will see the full MATLAB GUI that will open:



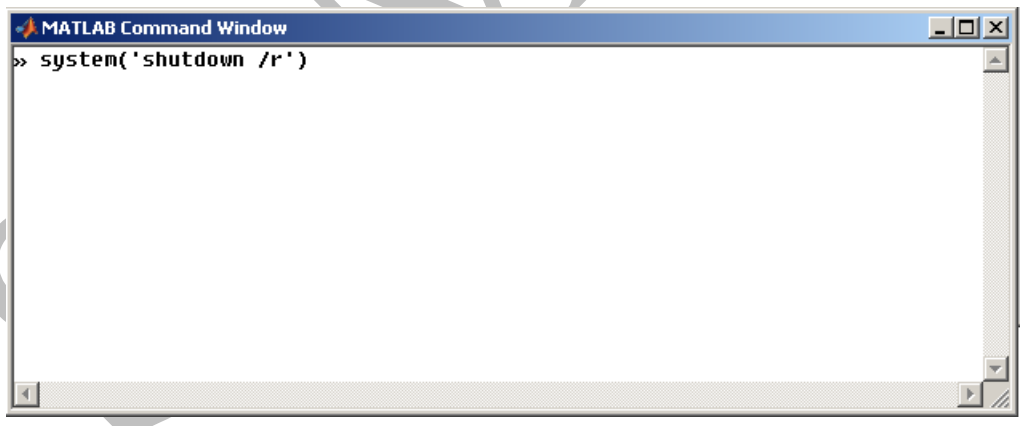
You can also close the Command Window or even the MATLAB GUI by typing **exit**:



From the Command Window you can type most MS-DOS commands if you write a exclamation mark **!** before the standard MS-DOS command:

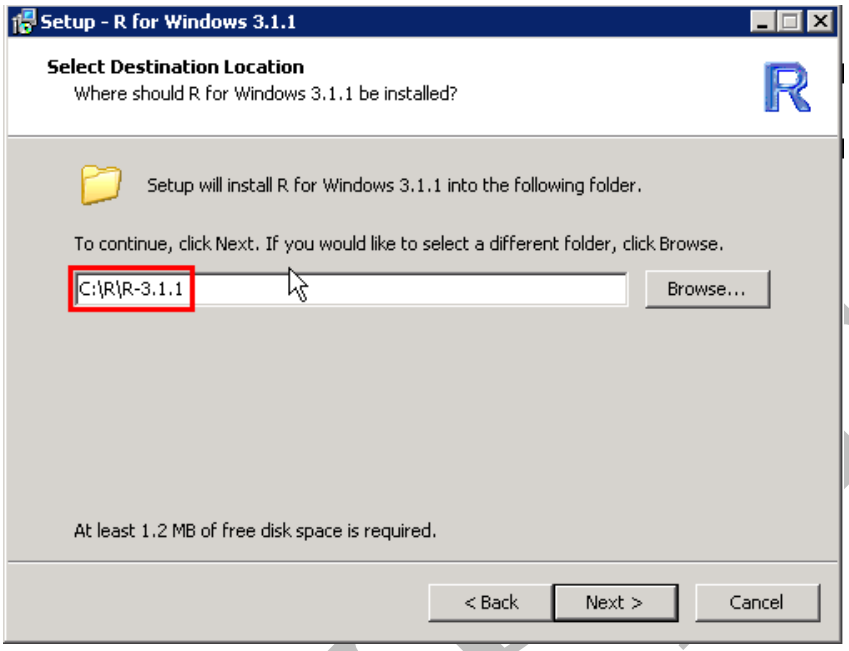


Or reboot the computer:



Call R from MATLAB

With such system commands you can call R software from MATLAB and retrieve results. For this purpose first install R:

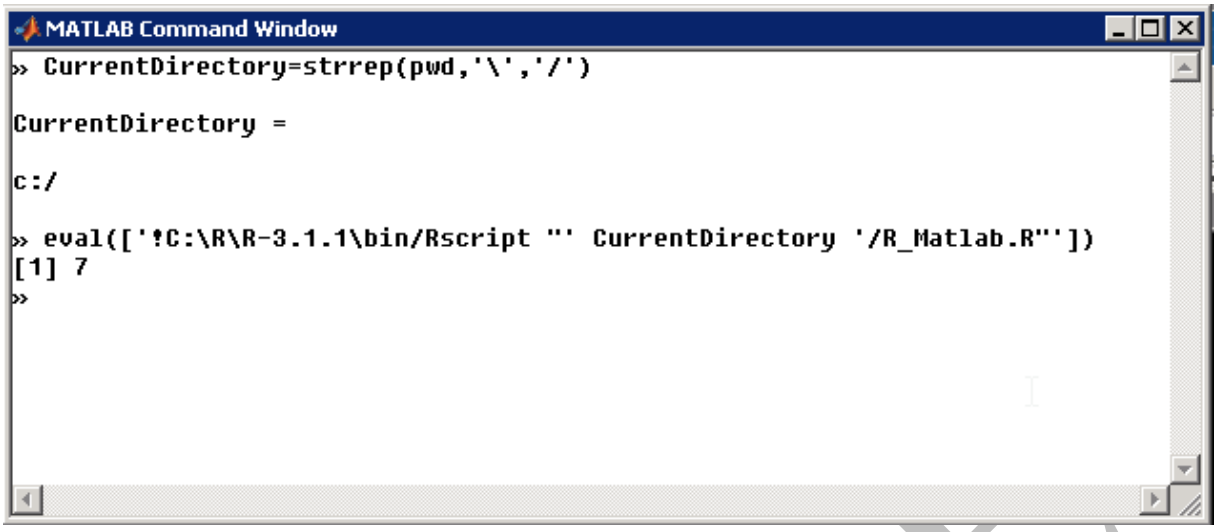


Avoid the usage of spaces in the installation folder!

One this is done create for example the following small *.R script and take care to save it (for this particular example) in the active MATLAB workpath:



Then you can call R from MATLAB simply by using the following command:


MATLAB Command Window
» CurrentDirectory=strrep(pwd,'\','/')
CurrentDirectory =
c:/
» eval(['!C:\R\R-3.1.1\bin/Rscript "' CurrentDirectory '/R_Matlab.R"'])
[1] 7
»

SAMPLE

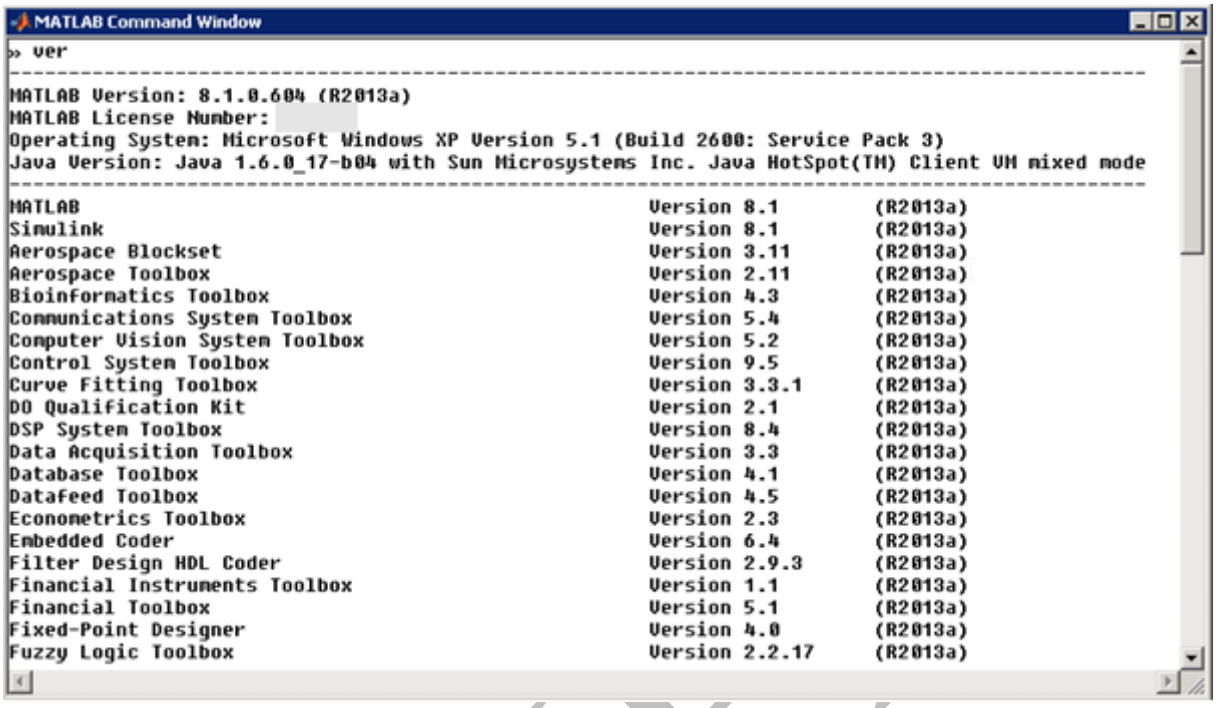
Get MATLAB system information

You can get a small resume of you MATLAB version by typing the command **version** and get the license number by typing the command **license**:

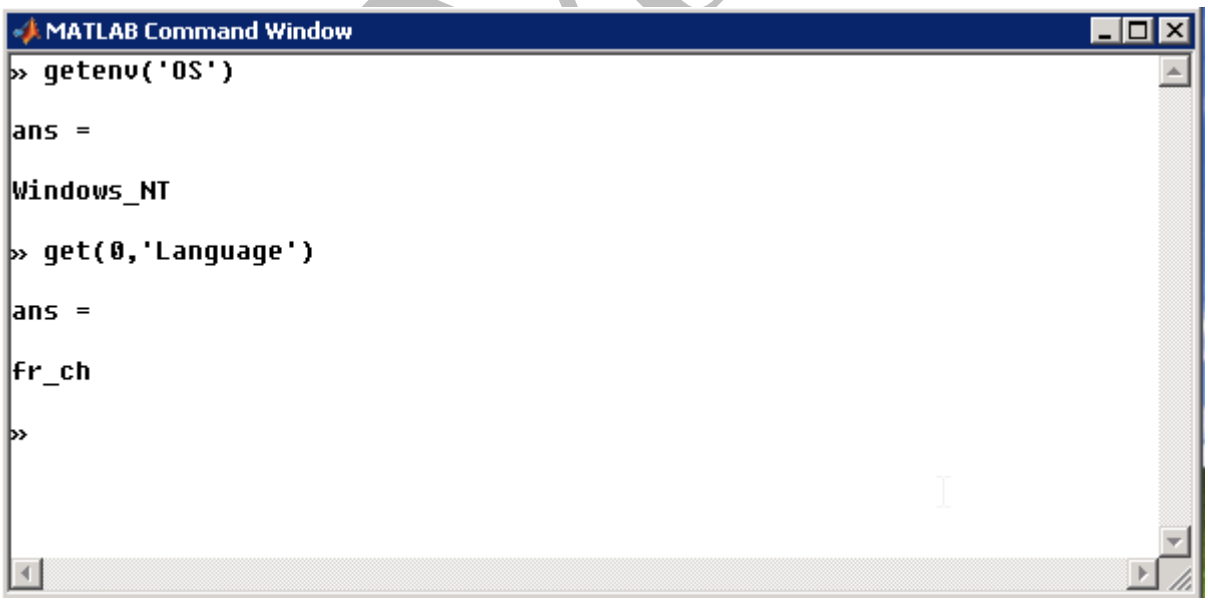


SAMPLE

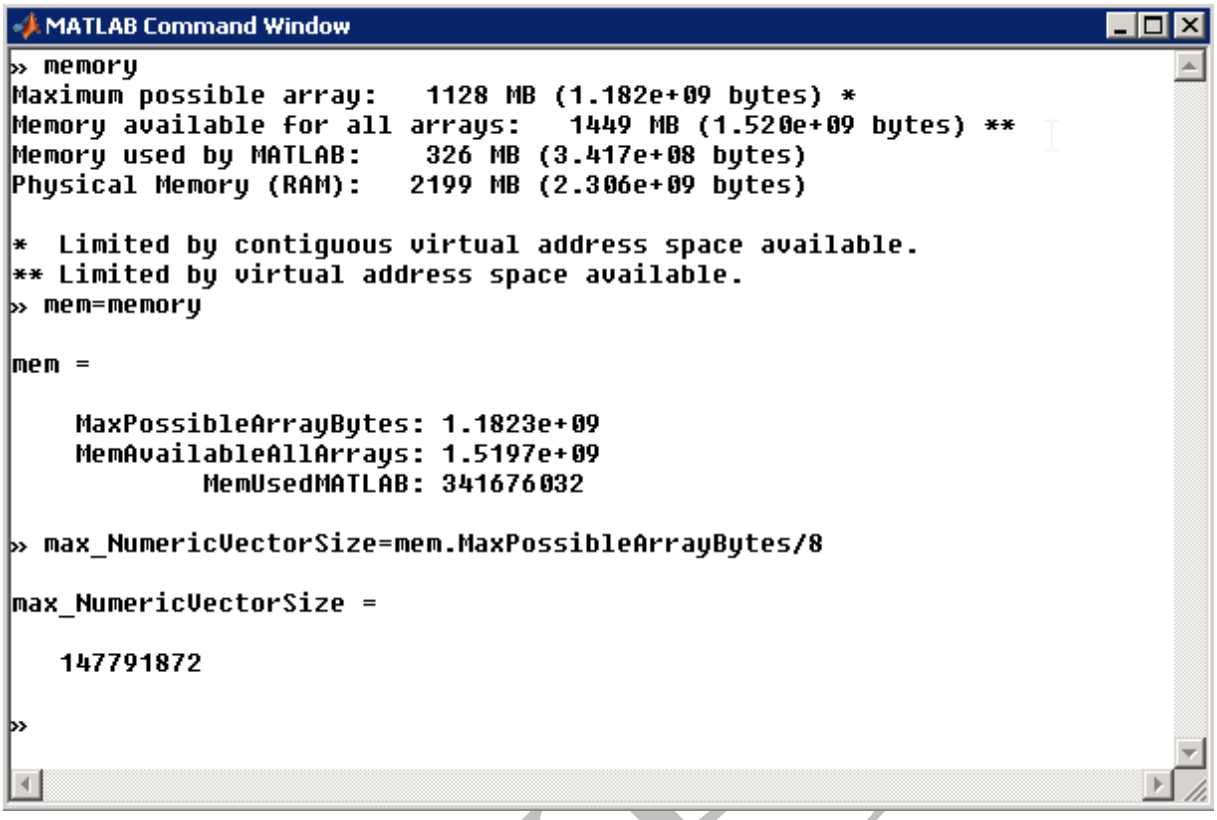
If you want to know the details of your version and the toolboxes available on your MATLAB installation you can use the command **ver**:



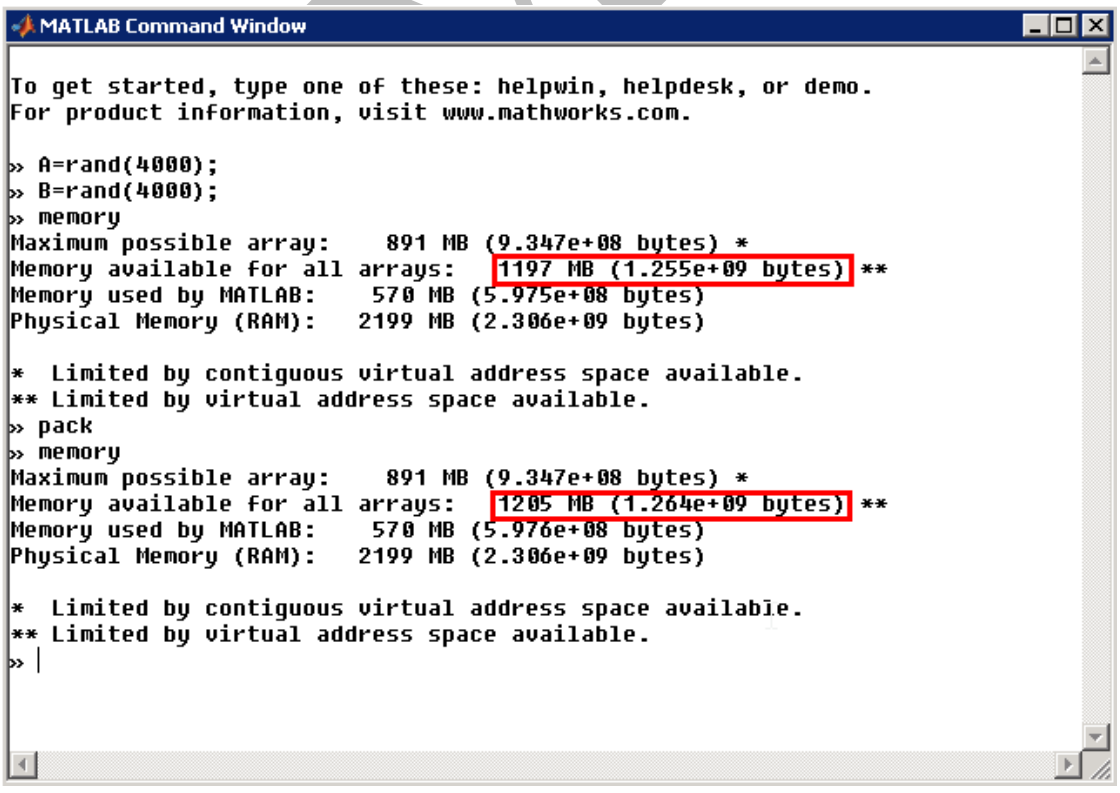
When you develop application it is also not only important to get the MATLAB version to ensure your script will run well but also the operating system name and language. You can get this informations by using the commands **getenv('OS')** and **get(0,'Language')**:



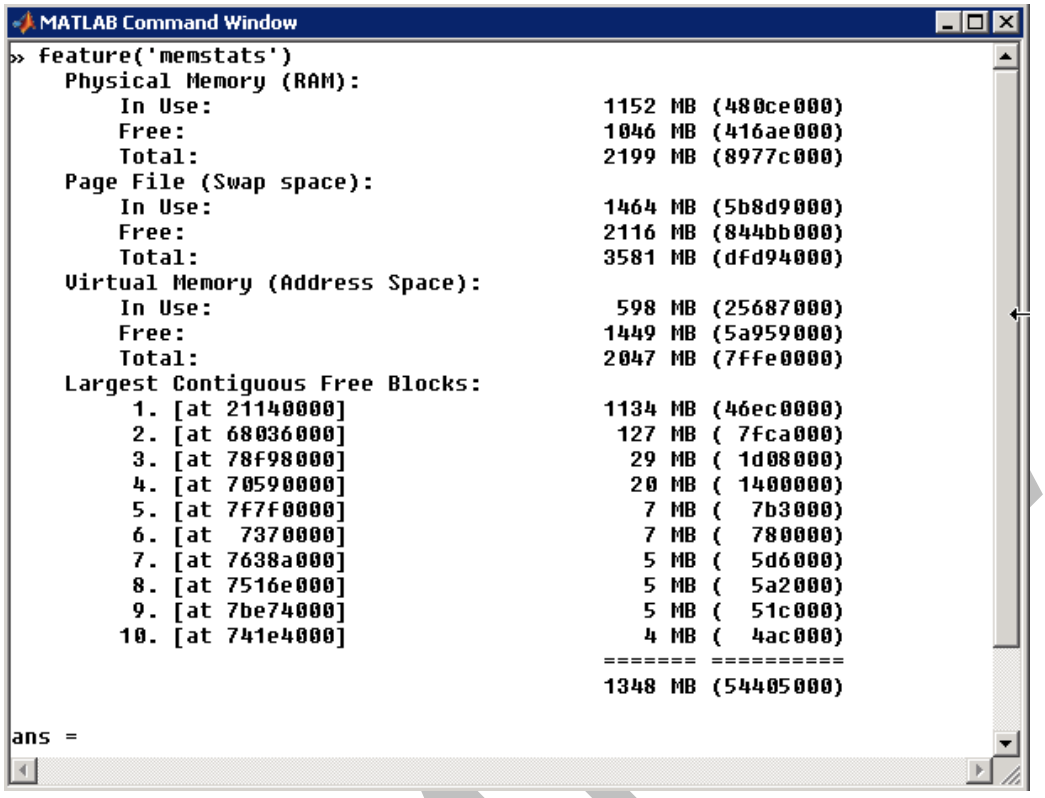
Another very important information that you need when you develop commercial information is the memory availabe and the biggest possible numeric vector size by using the method **memory** and its **MaxPossibleArrayBytes** property:



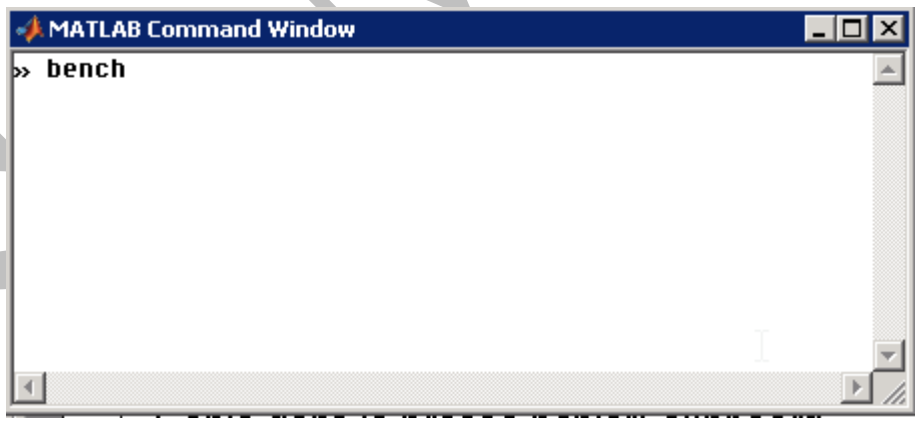
There is a command name **pack** that cannot be used in scripts files but that allows you to defragment the memory. In this way you will clean the memory a little bit and increase also a little bit the execution time of your scripts:



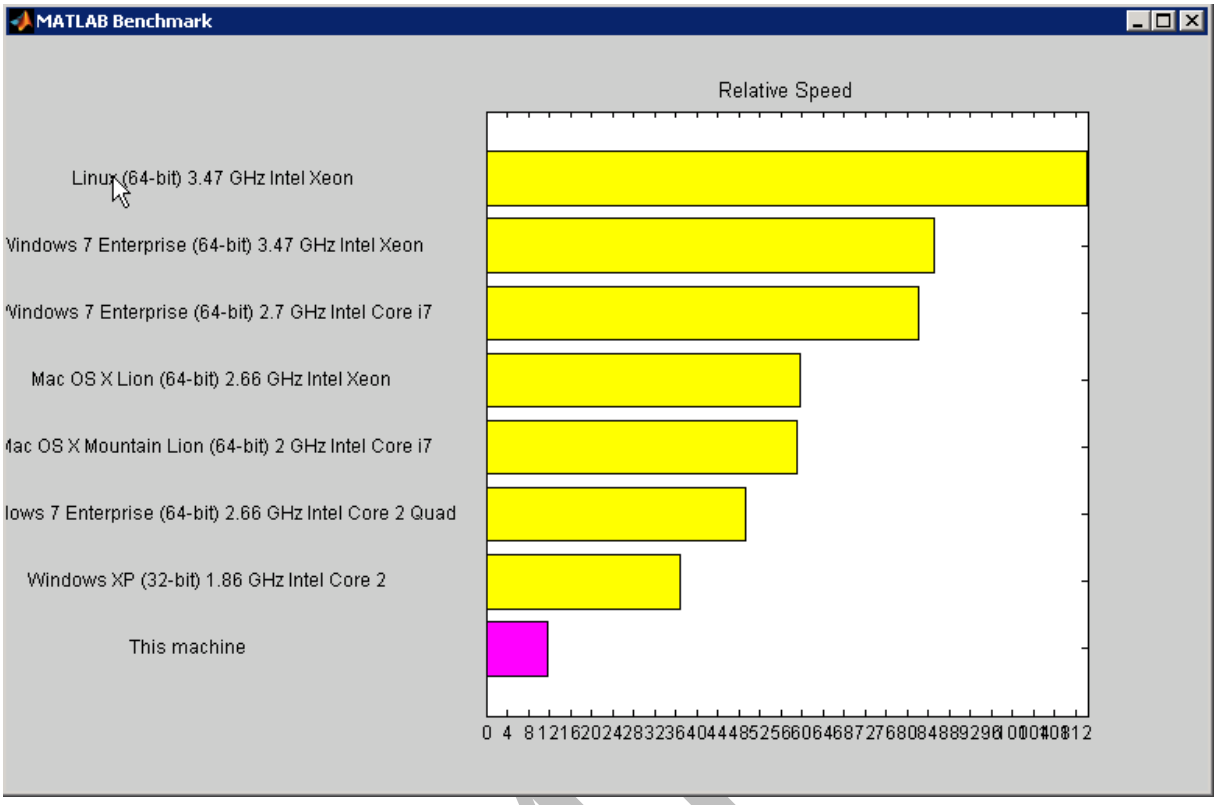
There is another way to see memory usage using the command **feature('memstats')**:



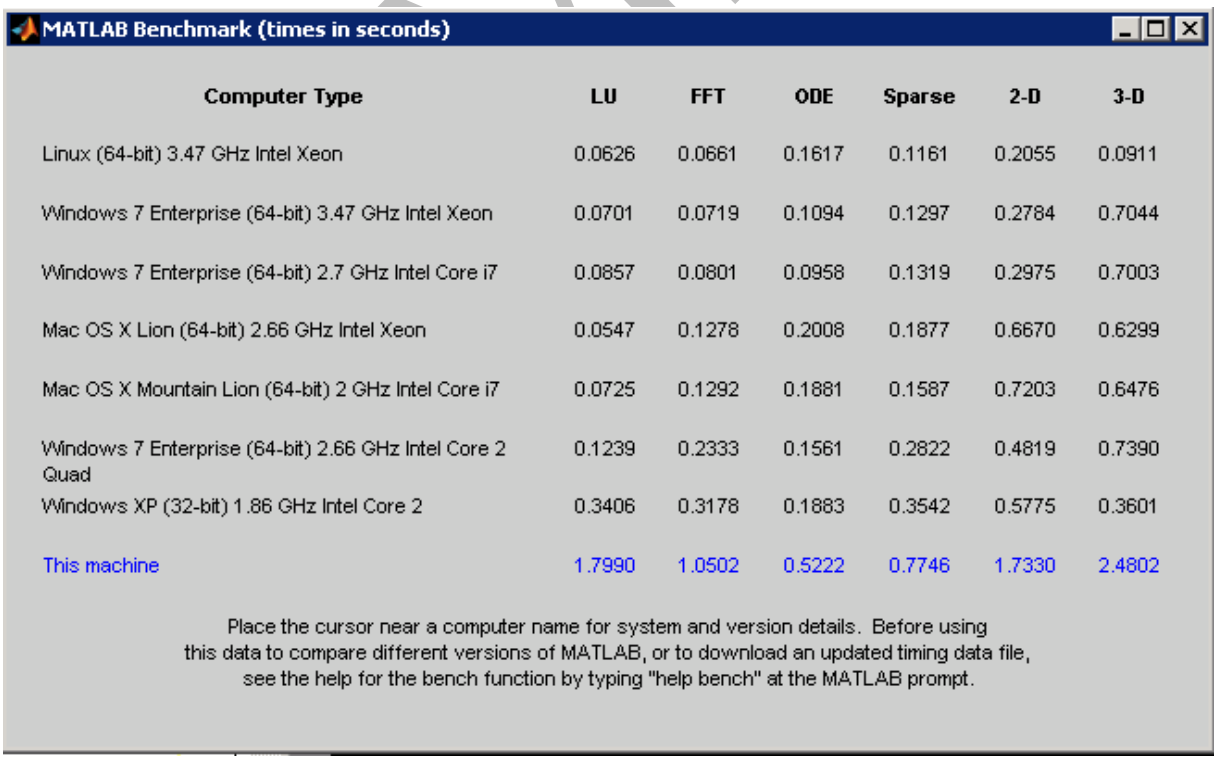
For fun or for technical support with a customer you can run the **bench** command to benchmark your computer performance with typical others systems (this test has been developed in association with creator of the old famous LINPACK super calculator megaflop test):



You will get a first chart after a few seconds:



You will also get a figure with a resume just after:

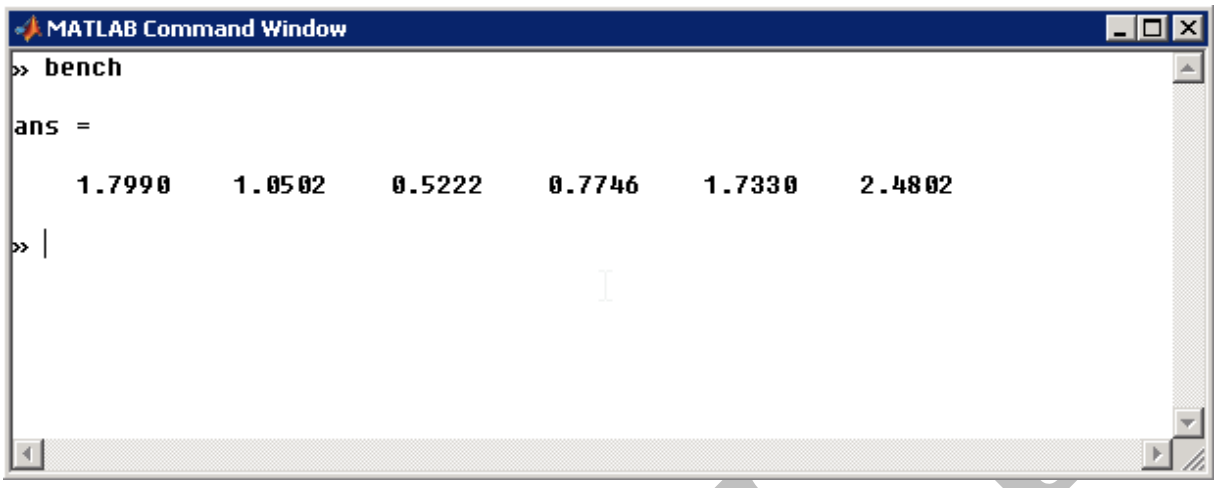


MATLAB Benchmark (times in seconds)

Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Linux (64-bit) 3.47 GHz Intel Xeon	0.0626	0.0661	0.1617	0.1161	0.2055	0.0911
Windows 7 Enterprise (64-bit) 3.47 GHz Intel Xeon	0.0701	0.0719	0.1094	0.1297	0.2784	0.7044
Windows 7 Enterprise (64-bit) 2.7 GHz Intel Core i7	0.0857	0.0801	0.0958	0.1319	0.2975	0.7003
Mac OS X Lion (64-bit) 2.66 GHz Intel Xeon	0.0547	0.1278	0.2008	0.1877	0.6670	0.6299
Mac OS X Mountain Lion (64-bit) 2 GHz Intel Core i7	0.0725	0.1292	0.1881	0.1587	0.7203	0.6476
Windows 7 Enterprise (64-bit) 2.66 GHz Intel Core 2 Quad	0.1239	0.2333	0.1561	0.2822	0.4819	0.7390
Windows XP (32-bit) 1.86 GHz Intel Core 2	0.3406	0.3178	0.1883	0.3542	0.5775	0.3601
This machine	1.7990	1.0502	0.5222	0.7746	1.7330	2.4802

Place the cursor near a computer name for system and version details. Before using this data to compare different versions of MATLAB, or to download an updated timing data file, see the help for the bench function by typing "help bench" at the MATLAB prompt.

and in the Command Windows you will see the above blue line available in the variable **ans** so you can manipulate these information later in your own scripts if you want to warn you customer about its low performance computer:

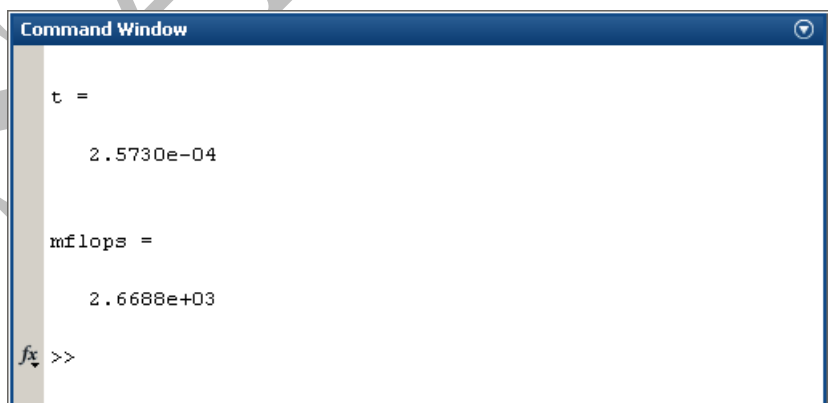


Remark: If you don't want the plots to appear, just edit the MATLAB script and comment the corresponding lines.

For people that want to reproduce the old 1997 LINPACK test (LINPACK in the name of the FORTRAN library for linear algebra) you just need to write in a script the resolution of a 100 by 100 linear system:

```
1    n=100;
2    A=randn(n,n);
3    b=randn(n,1);
4    tic
5    x=A\b;
6    t=toc
7    mflops=(2/3*n^3+2*n^2)/t/1.e6
```

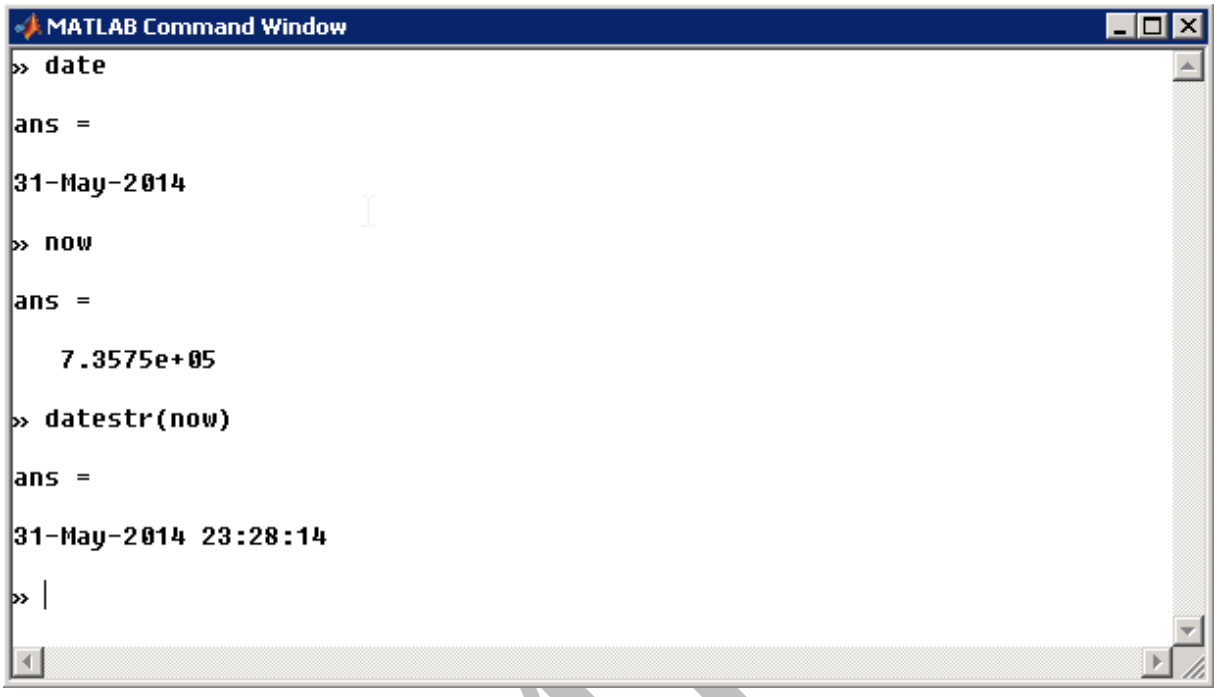
and we get:



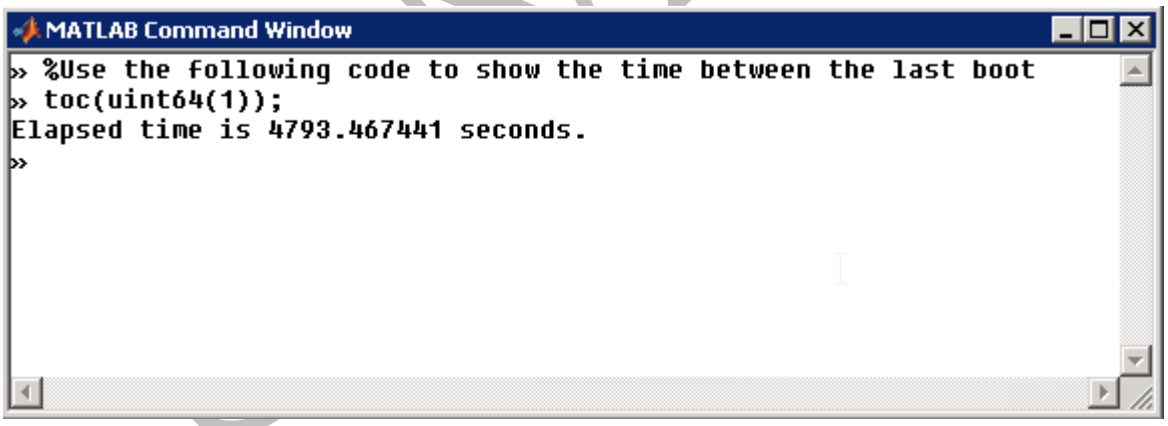
The n^3 term in the numerator of the last expression is roughly the number of floating point additions and multiplications required to compute A\b. The 1.e6 term in the denominator convers flops to megaflops.

Some dates stuff

Late we will study dates in details but we can also consider that some dates manipulations command are very general. This is the case of the **date**, **now** and **datestr(now)** commands

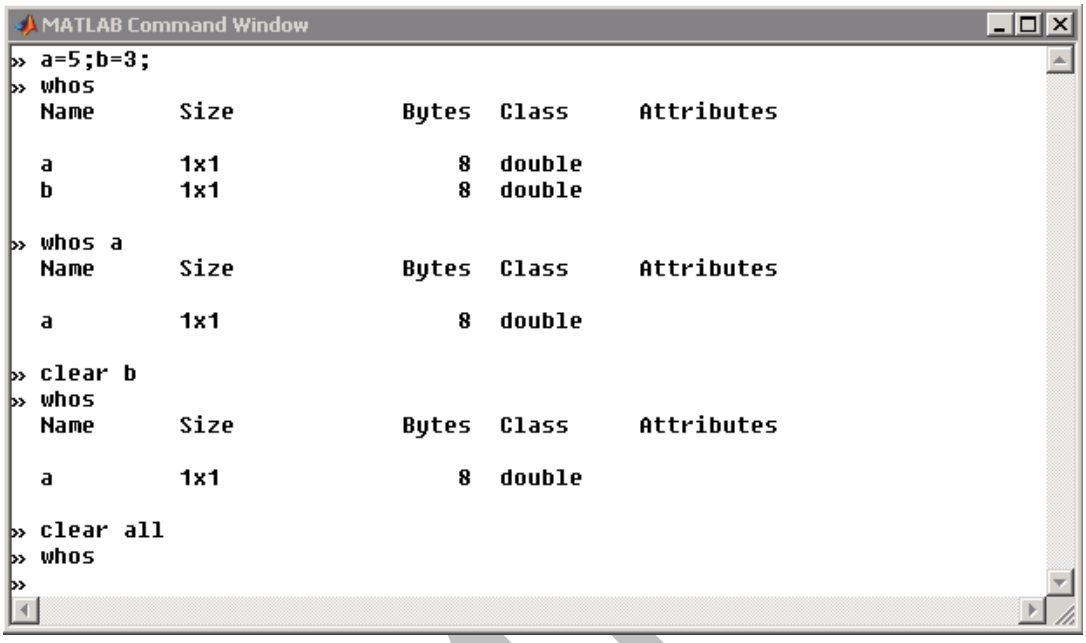


Sometimes your customer will forget to periodically reboot its servers or computer. To check since how many seconds the computer is started you can use the command **toc(uint64(1))**:

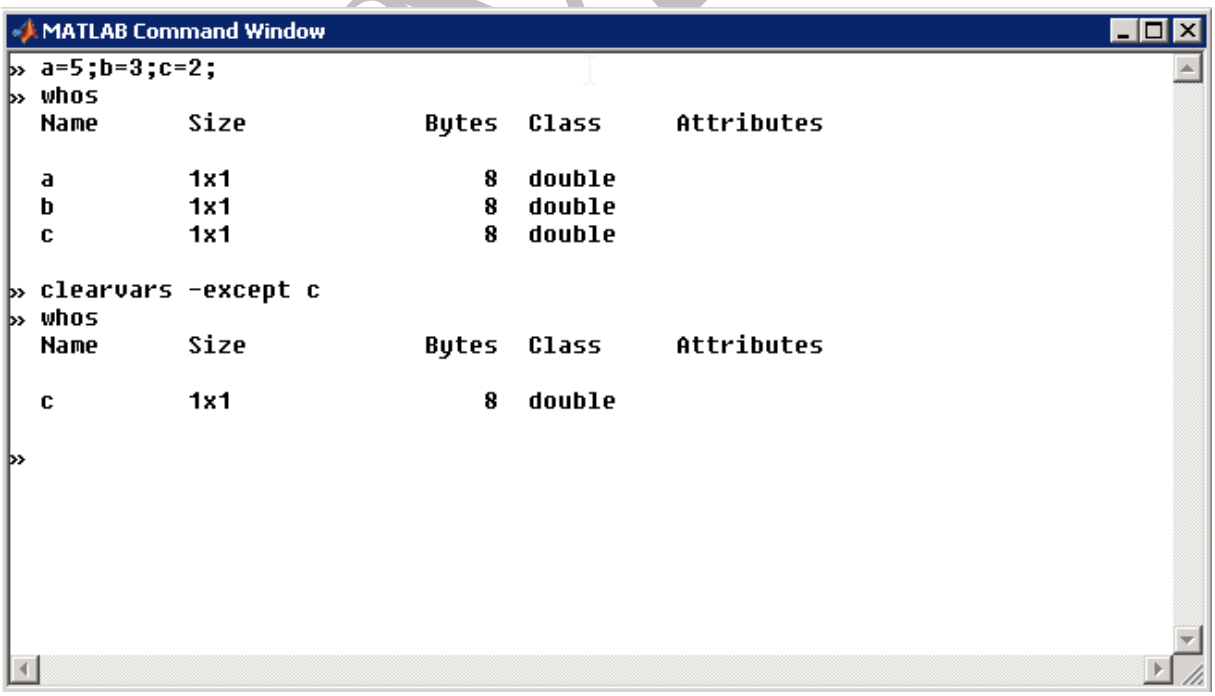


Manage variables

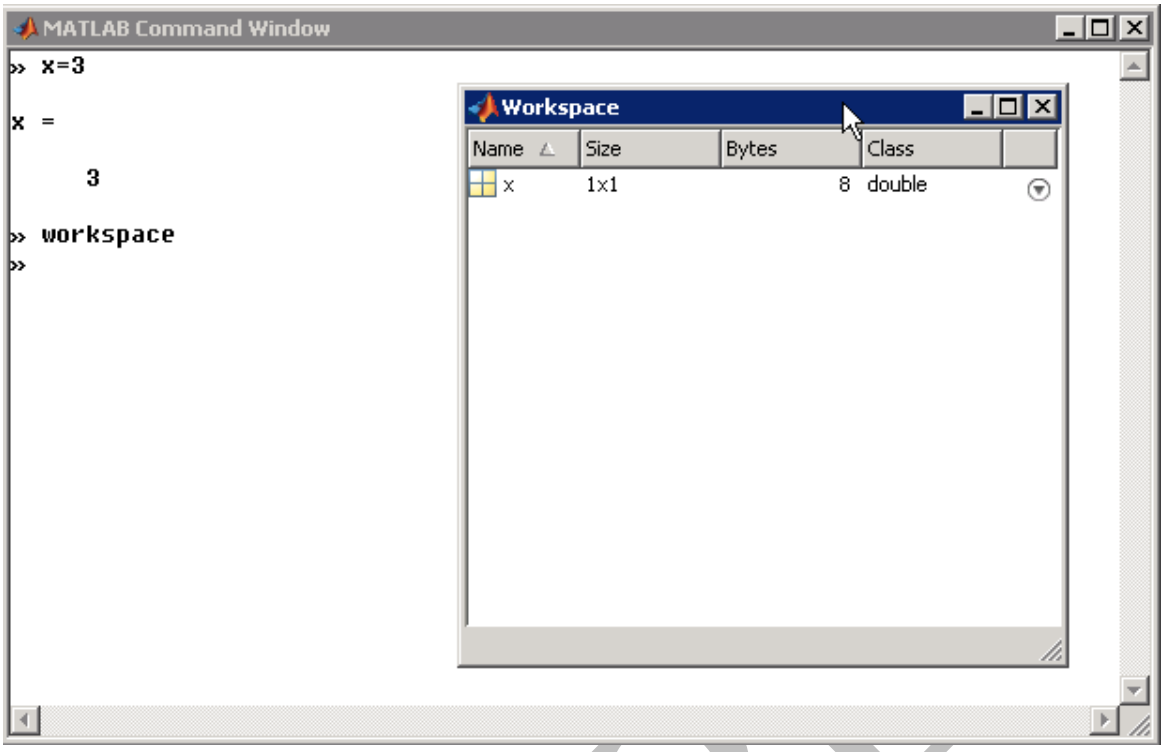
If you create variables in MATLAB and use the command **whos** you will have information about all variables or only some variables of the current workspace. If you use **clear** or **clear all** you can remove one or more variables:



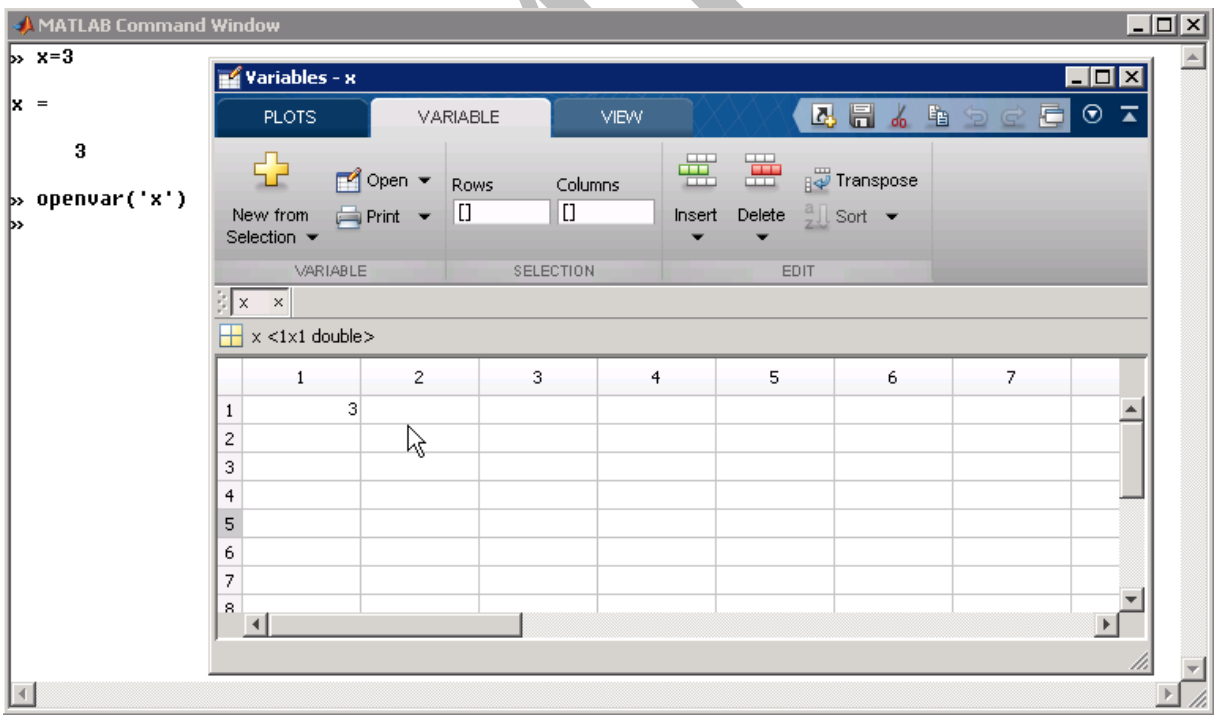
Sometimes when running script you want to delete all variables and keep only the final one. For this you can use the command **clearvars –except**:



You can also see the variables workspace windows by typing the command **workspace**:

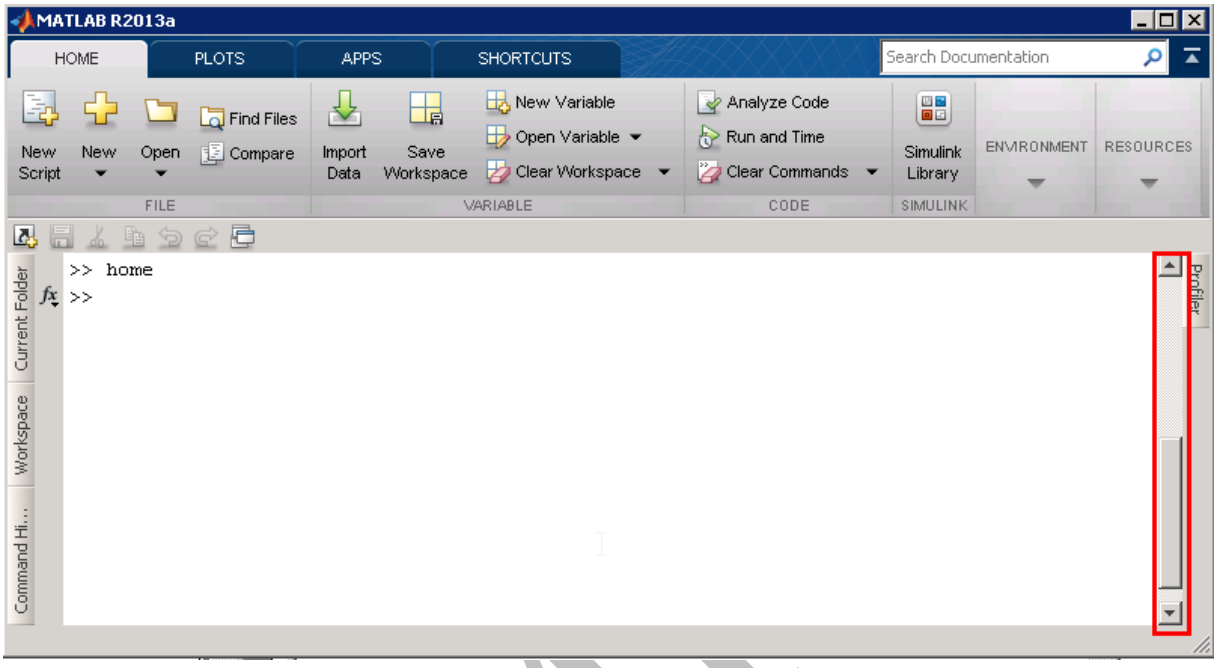


And if you have complex variables (arrays, cell array, dataset...) you can edit some of them by using the command **openvar** and you will have the variable editor that will open.

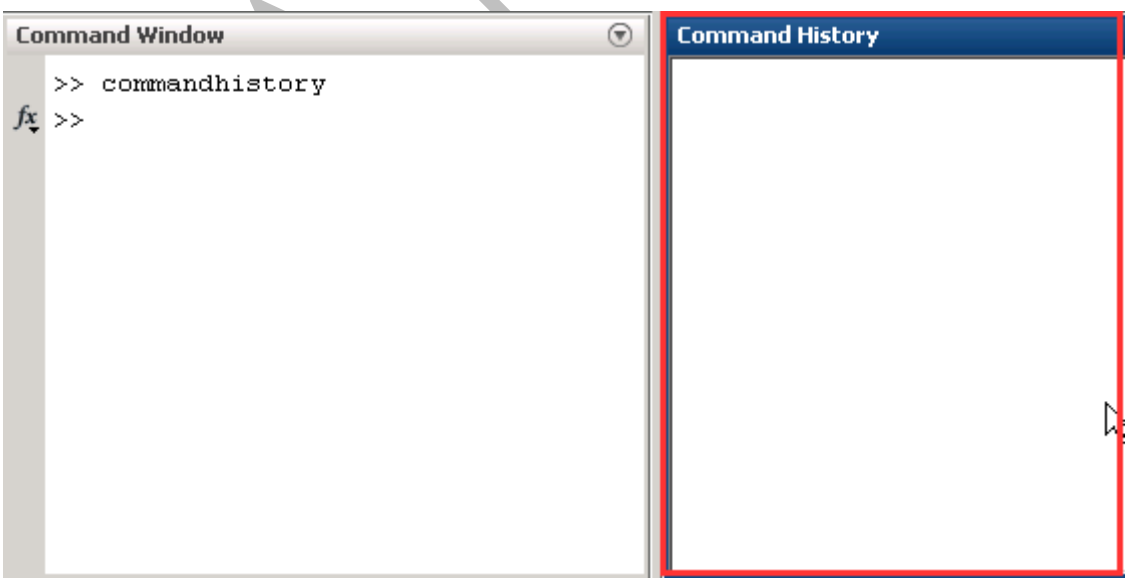


Some Command Window controls

If you are using the MATLAB GUI by typing the command **home** you will get back to the top of the Command Windows (in the old style Command Windows this will just have the same effect as using the command **clc** that clear all previous written commands):



If you type **commandhistory** in the MATLAB GUI this will force the the Command History window to appear (**the *Command History* windows shows only the commands that were typed in the MATLAB GUI not the commands typed in the old style MATLAB Command Window!!!**):



If you want to clear the content of the history without using the mouse but with a simple command you will have to use the following command:

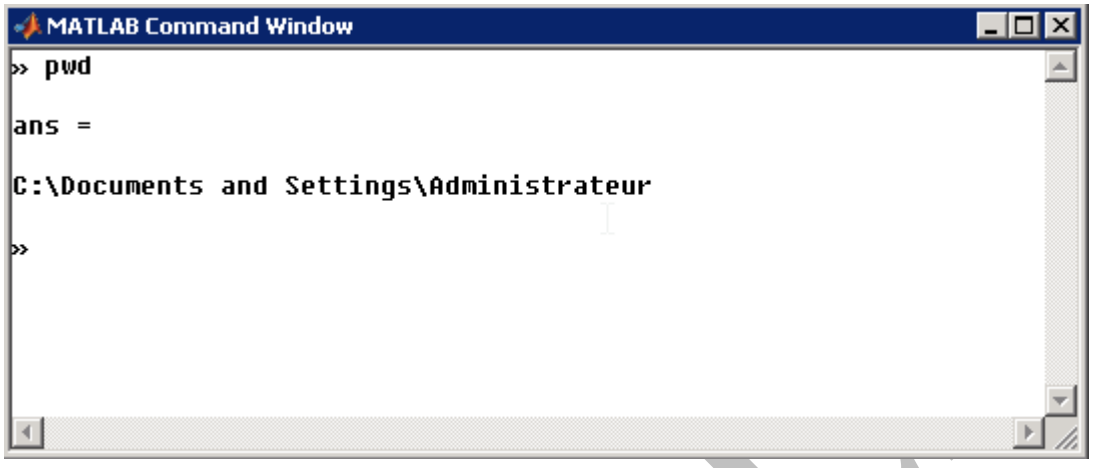


Com.mathworks.mlservices.MLCommandHistoryServices.removeAll

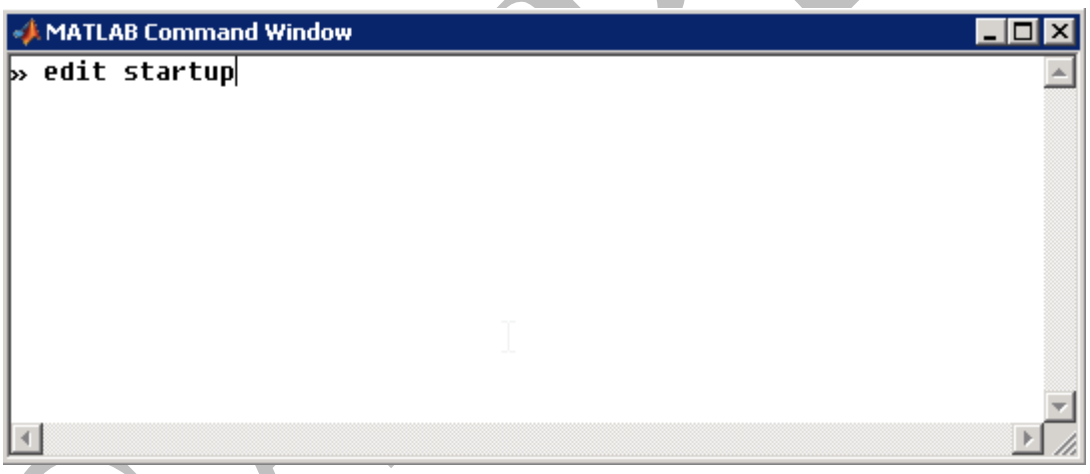
SAMPLE

Use the startup.m file

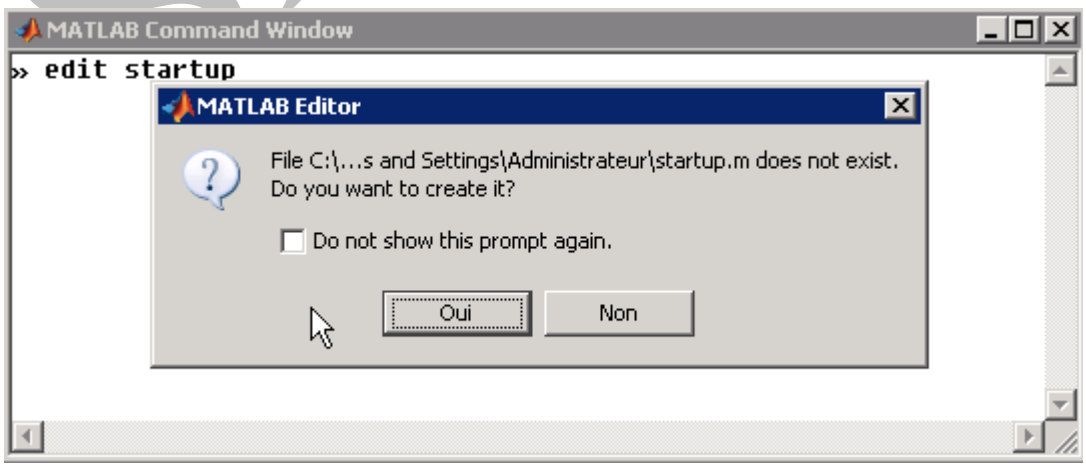
By default MATLAB has a workspace path where you can put a *startup.m* file script that will run each time you will open the software. You can get this path by typing **pwd** (we will see later how to change it):



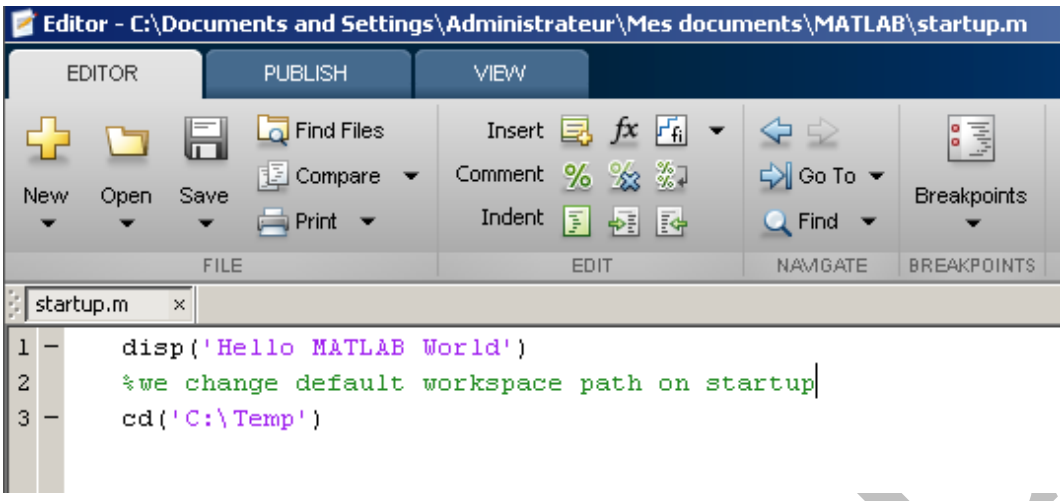
After if you want to create the *startup.m* file for the first time just use the command **edit**:



If the file does not exist, MATLAB will ask you if you want to create it:



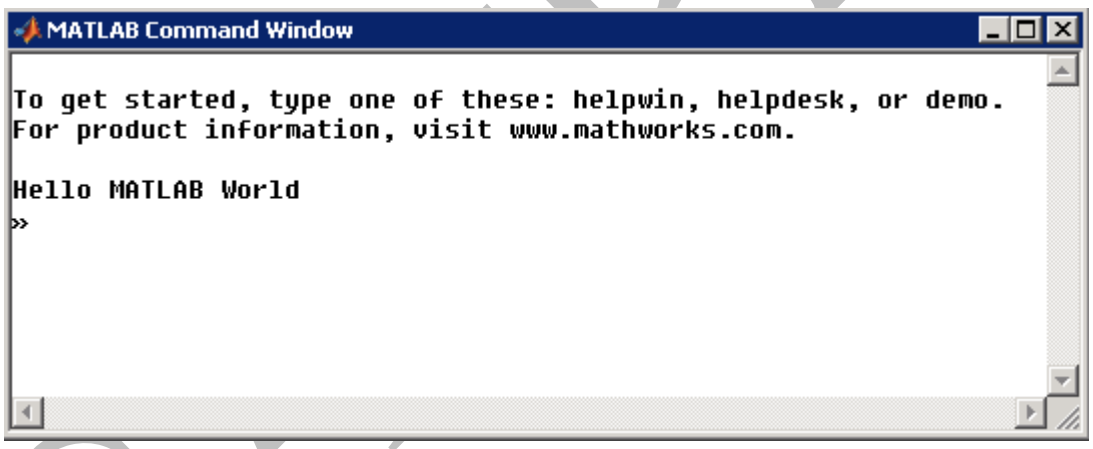
Then the *startup.m* file will open in the MATLAB Script Editor:



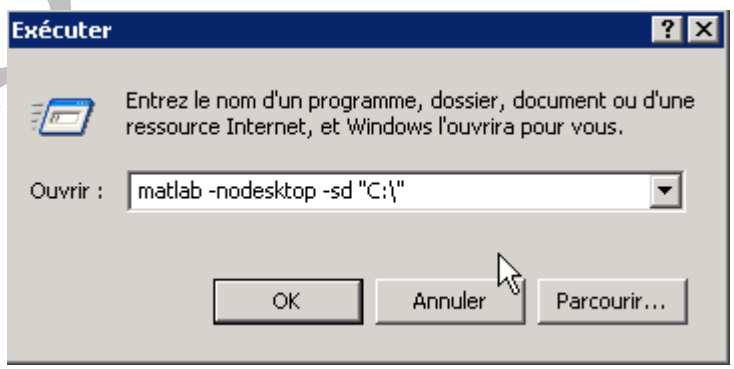
You can type a lot of stuffs in the *startup.m* file...

Remember: You can't put the *pack* command in the *startup.m* or any other script!!!

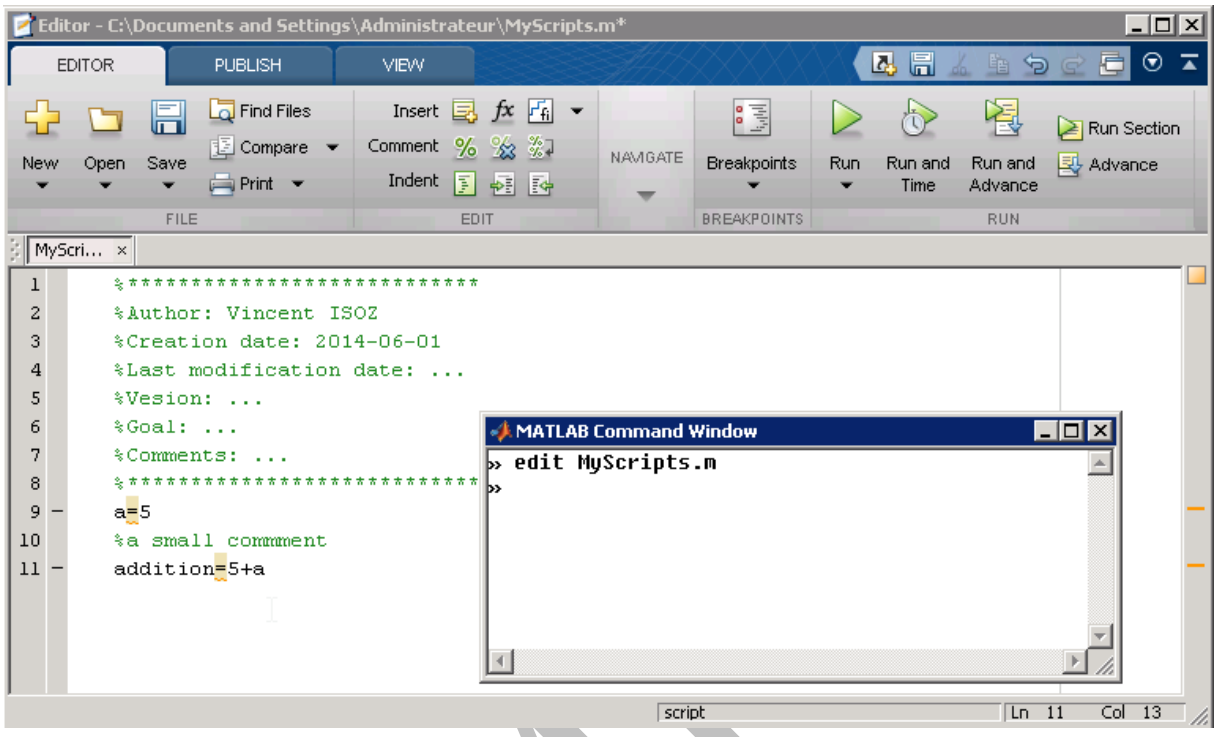
If you save the file and close/reopen MATLAB you will get:



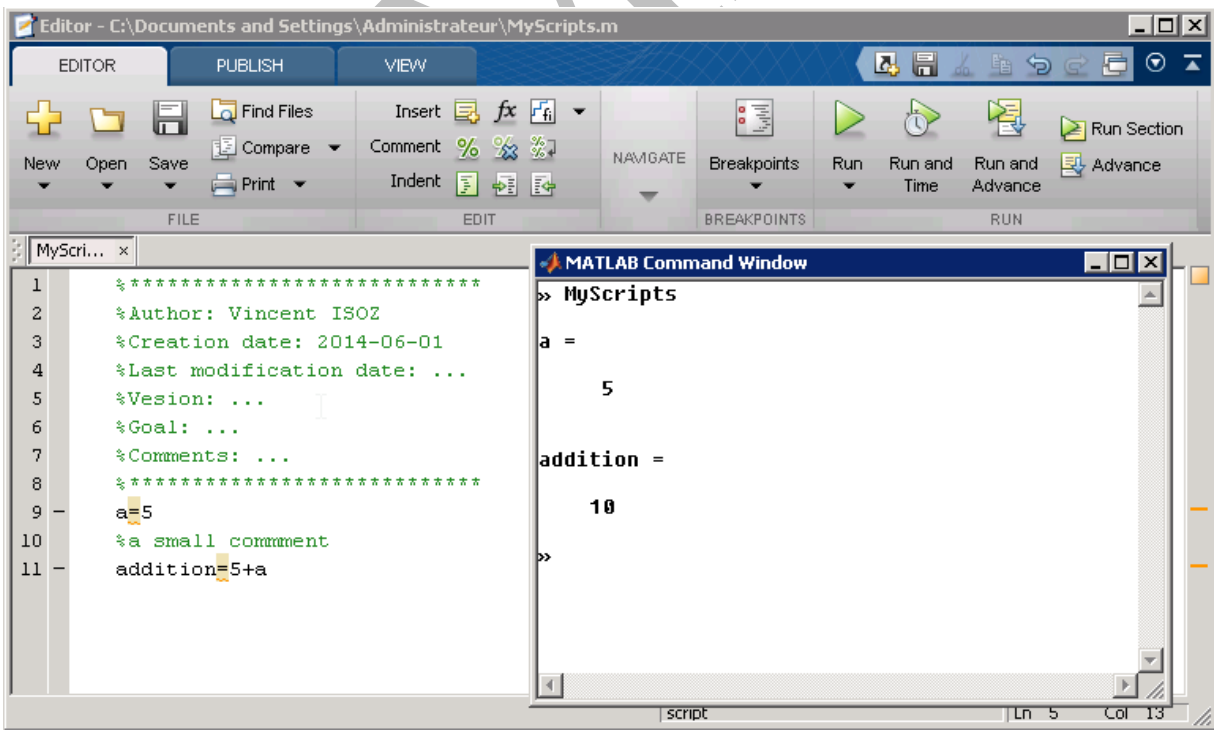
If you want to change the MATLAB default start directory (**sd**) you can change the icon shortcut with the following options:



In the facts you can create and use any *.m script file following the same process. For example if we want to create in the current working path a new file named *myScripts.m* we will write the following:

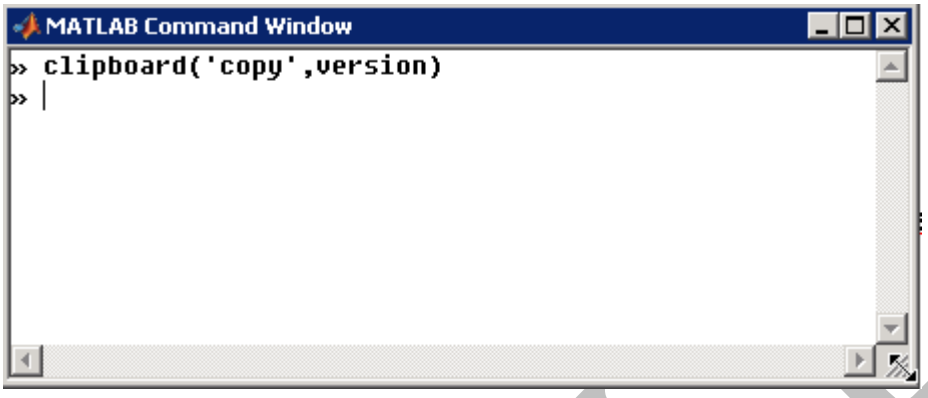


And to run it we will just write the script file name:

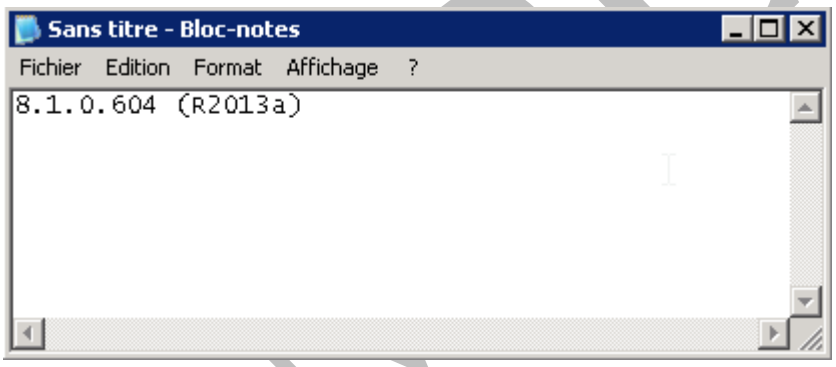


Use the clipboard

You can copy with de command **clipboard** MATLAB information or variables so the user will then be able to make a simple paste in its favorite application (you must have you favorite application already opened before you launch the coyp command):



and if we open the Notepad for example and we paste (Ctrl+V) we get:



SAMPLE

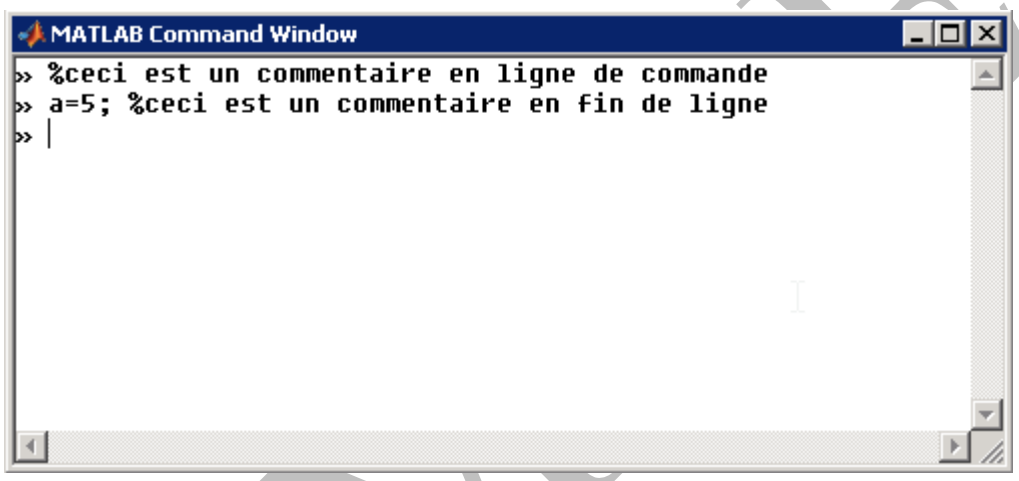
Comments and multiple line commands

Typing comments is very important in any computer language. In MATLAB there are three type of comments (we will see them all later): simple line comment that begin with **%,** simple line title comments that begin with **%%**, and multiple line comments:

```
%{
...
}%
```

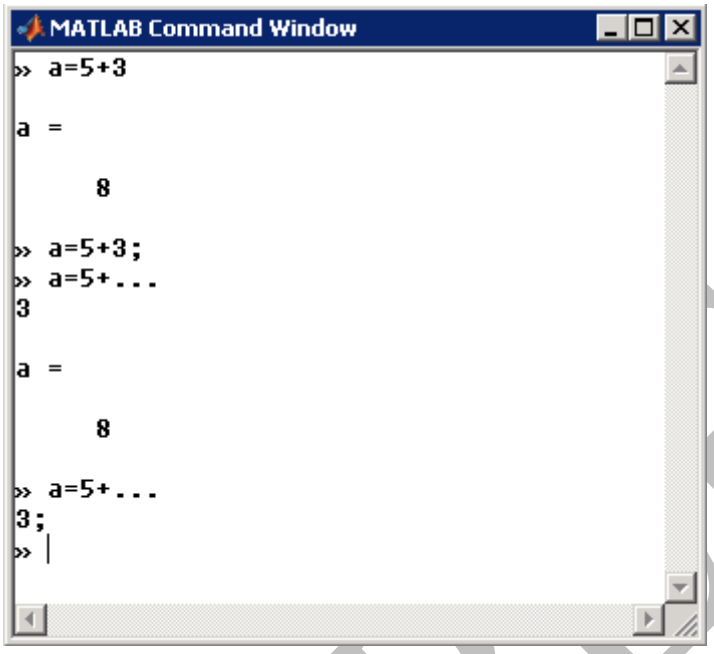
as you can see on page 543.

You can use simple comments in the shell also:



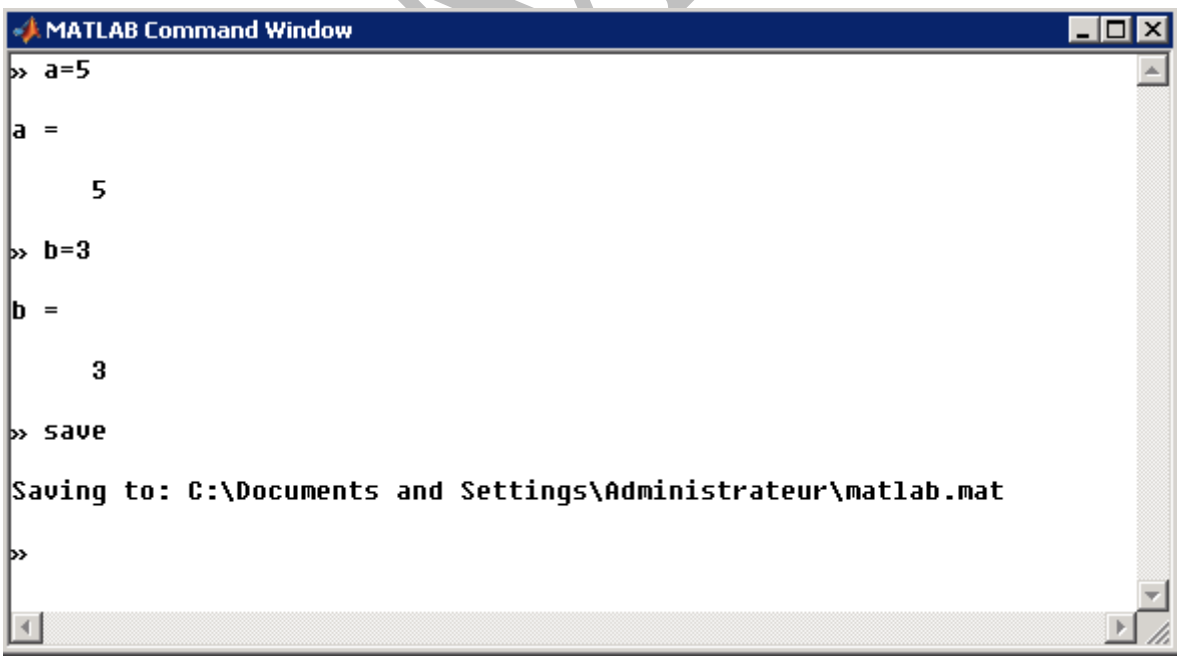
SAMPLE

Many times in MATLAB you will have to type very long command. To cut on command on multiple lines just type three dots ... just after a comma or a mathematical operator:

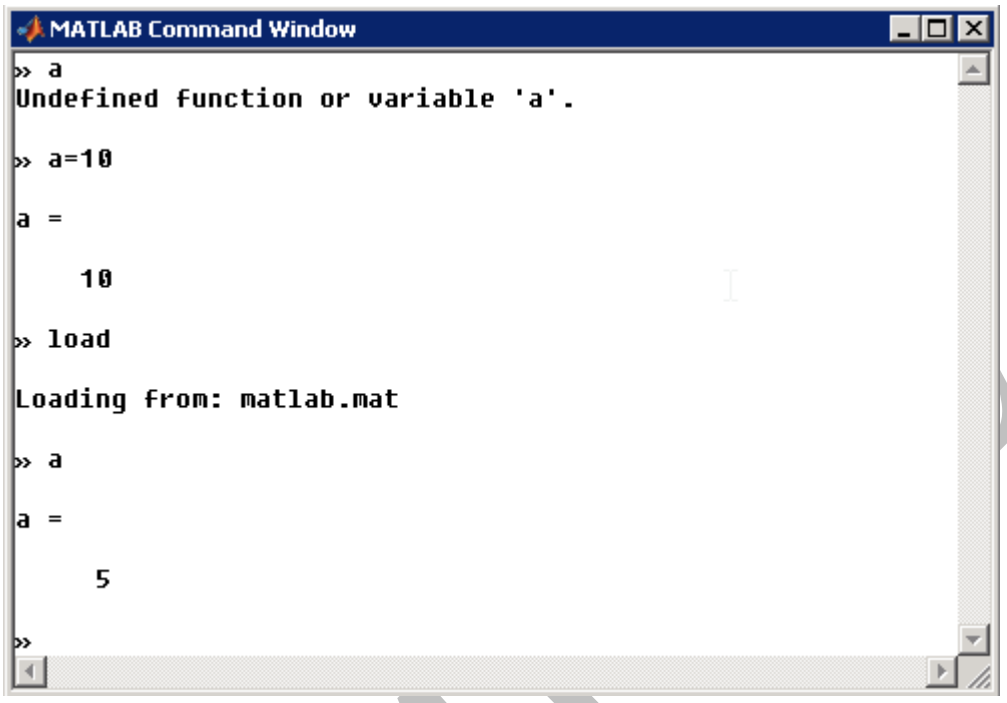


Save workspace/Use commands diary

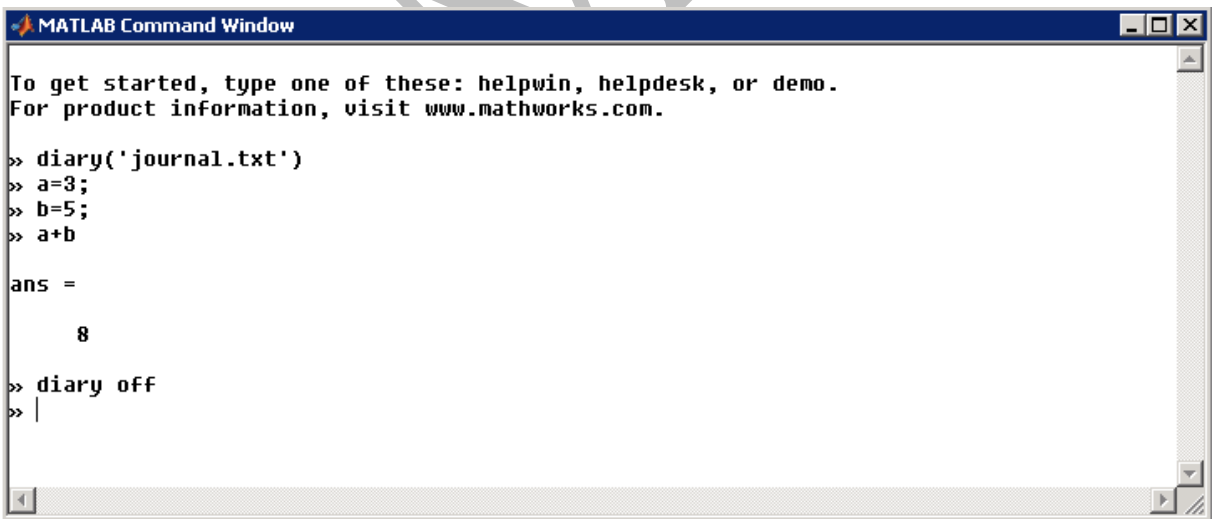
You can save all variables you have created or imported in a binary *.mat file to recover this informations later. Using the command **save**, you will save all the variables of the workspace by default in the current working path under the default matlab.mat filename:



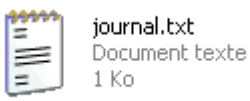
If you make a **clear all** and try to access to one of your previous variable, it will be empty but if you load the previous matlab.mat file using the **load** command the variable will be again accessible:



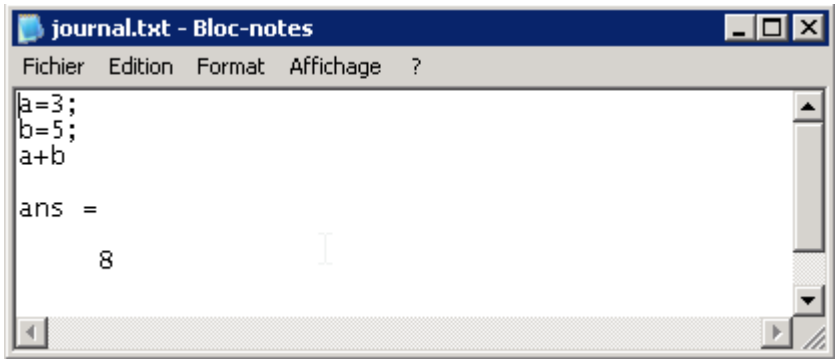
If you want you can ask MATLAB to save in a text file all comands that you type. For this you just have use the command **diary** and specify between bracket the path and filename you want to use for this purpose:



Then you will get a new file on your hard drive:



The content will looks like following:

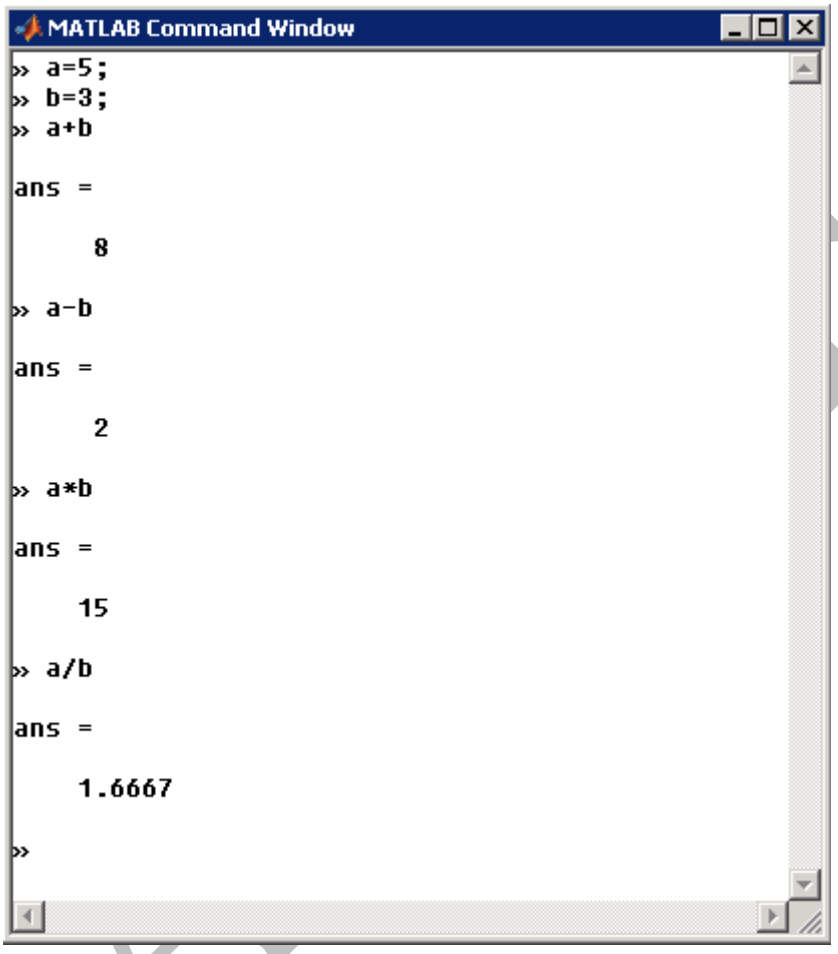

journal.txt - Bloc-notes
Fichier Edition Format Affichage ?
a=3;
b=5;
a+b
ans =
8

SAMPLE

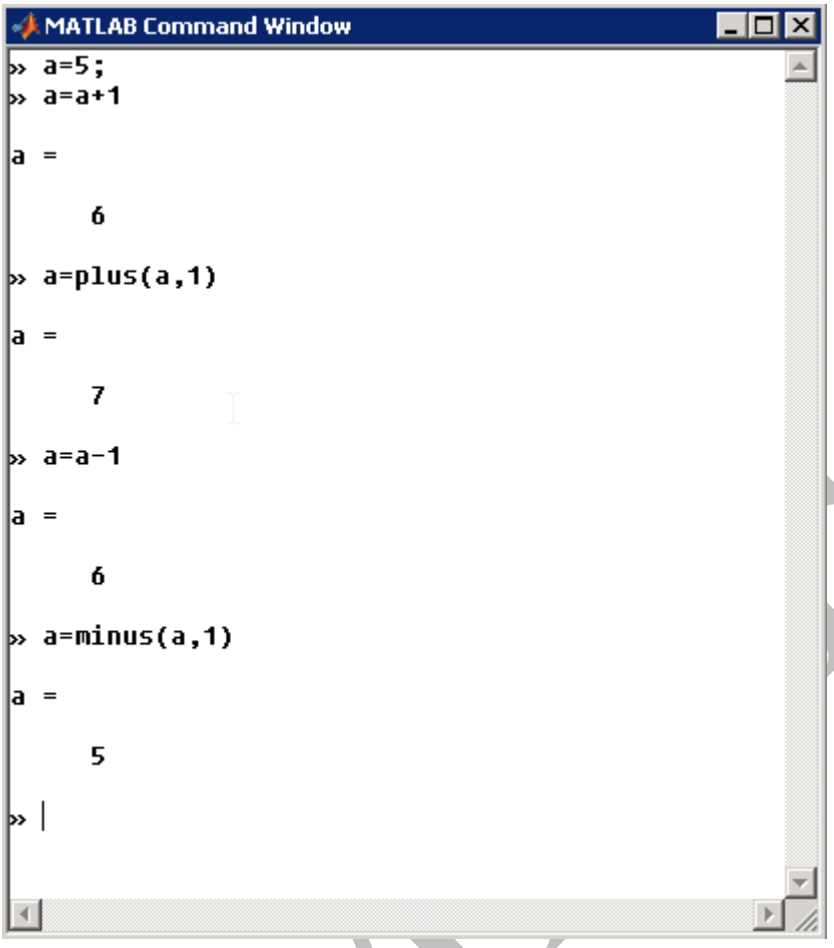
Arithmetic operations on scalars

The purpose of this chapter is to have a look to the syntax of very trivial scalar number arithmetic operators in MATLAB.

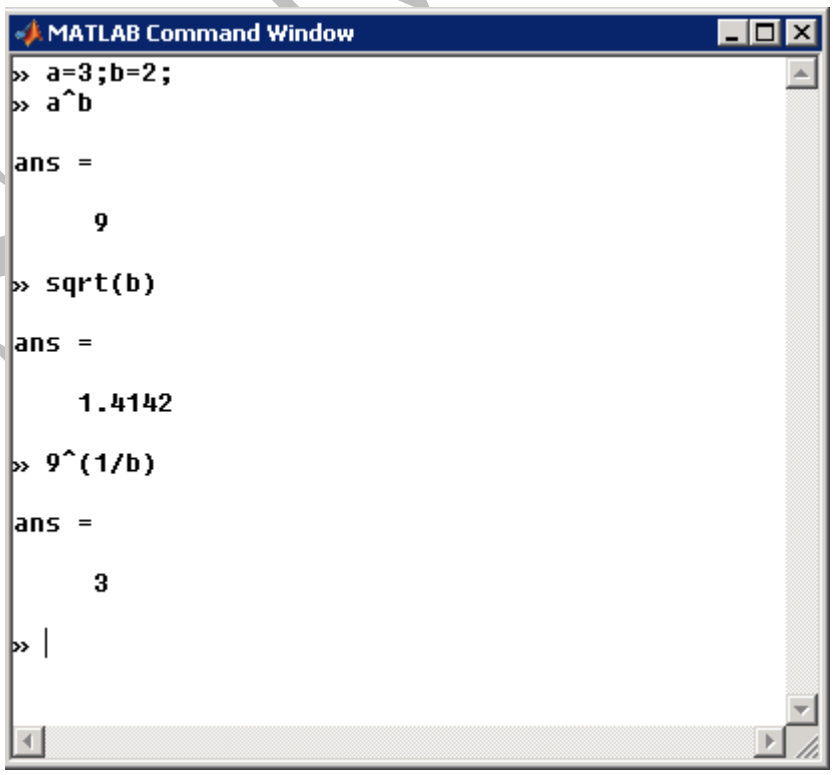
We first begin with addition, subtraction, multiplication and division:



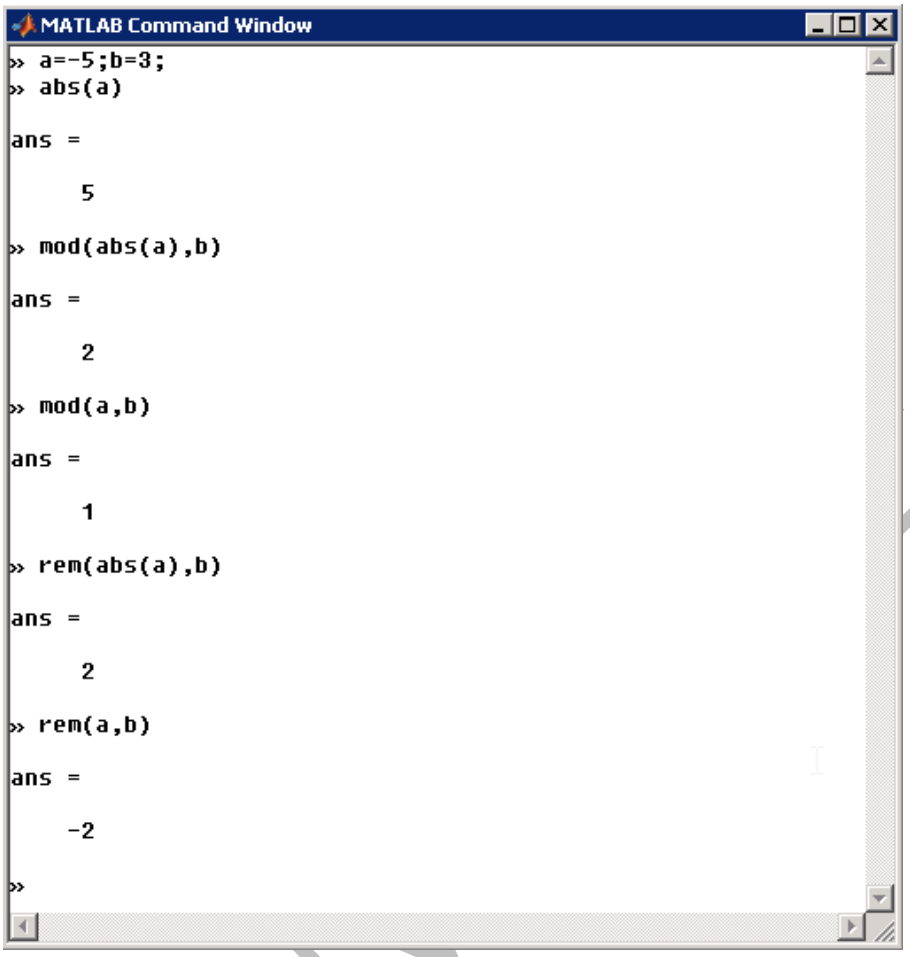
There is also another way to add/substract in MATLAB using the functions **plus( )** and **minus( )** that is use sometimes by MATLAB users but used a lot when you want to optimize matrices addition/subtraction as we will see it later:



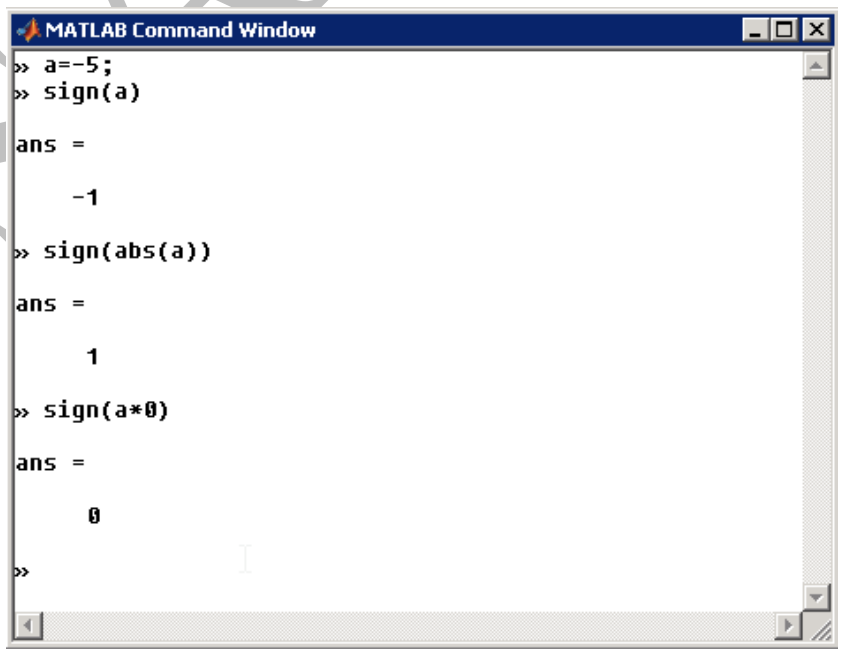
Now we take a look to powers and roots (as you can see the square too can be obtained with the classical **sqrt( )** function):



Now we take a look the classic absolute, modulo and remaining calculations whose respective functions are **abs( )**, **mod( )** and **rem( )**:

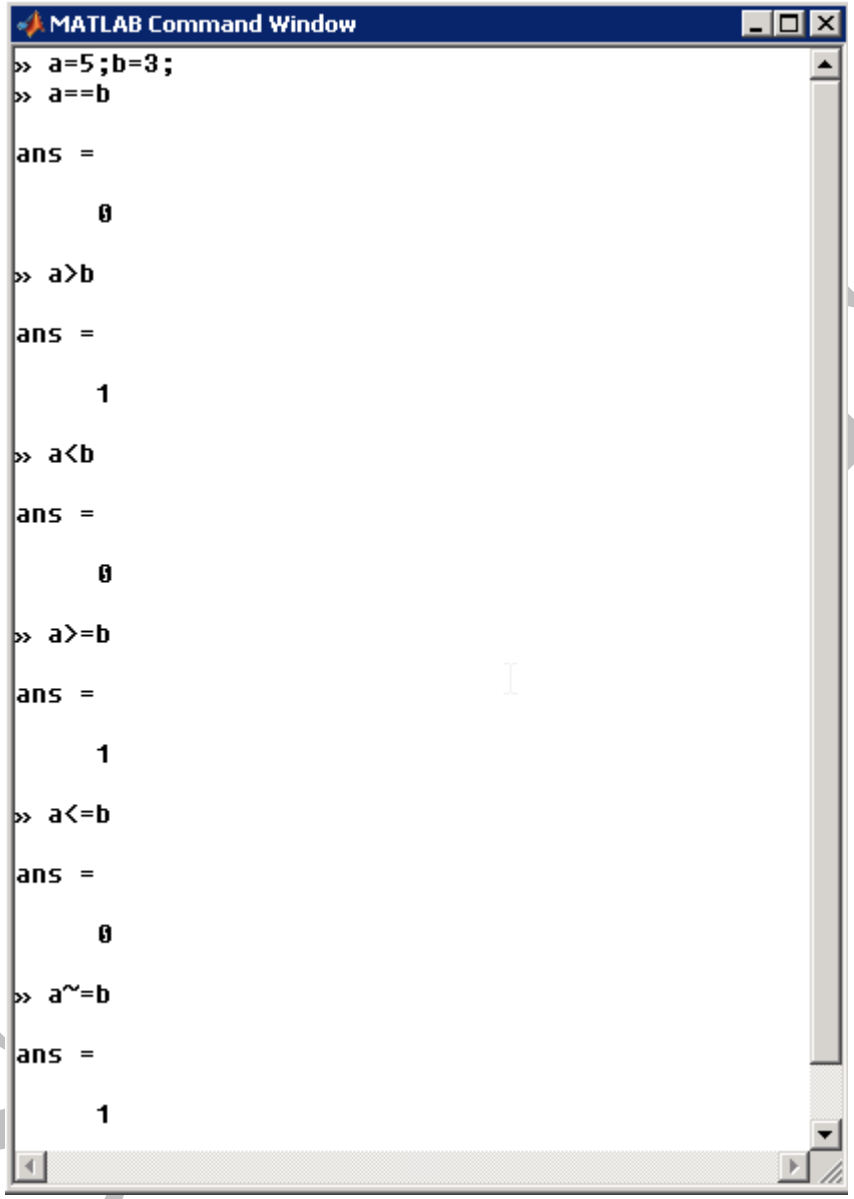


We also have the very classic and sometimes useful **signum( )** function to get the sign of a function:

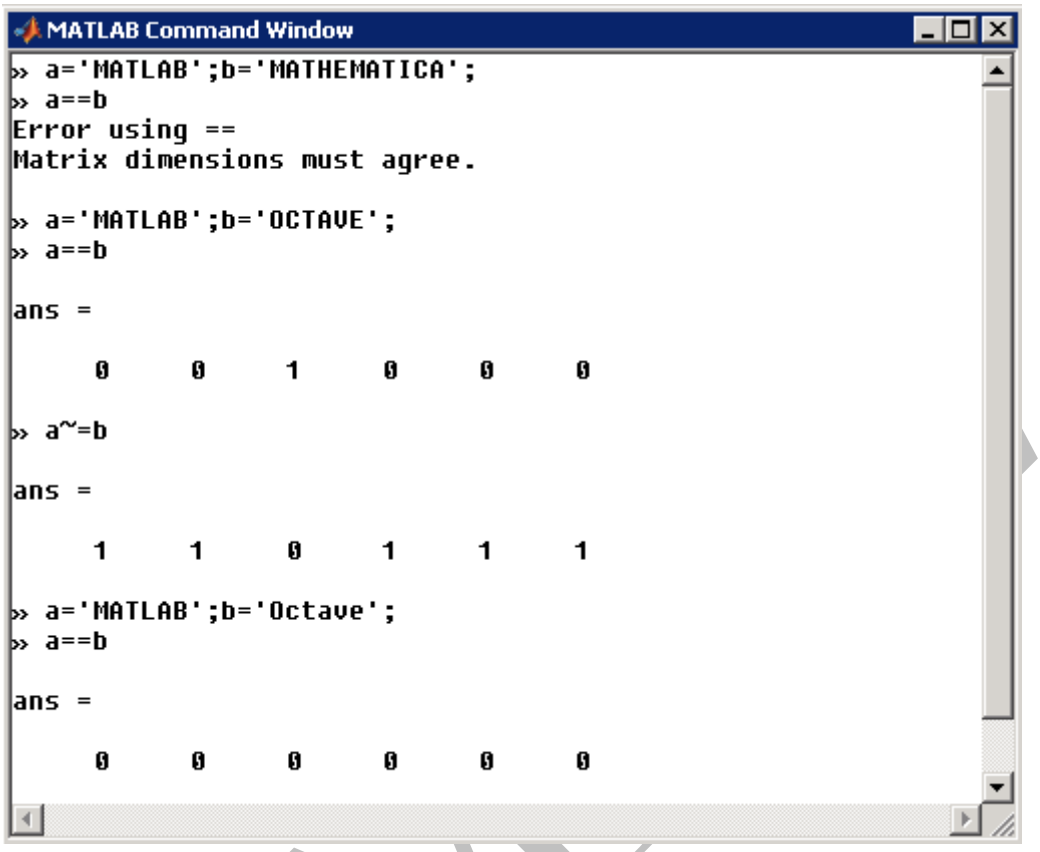


Operators of comparison

All scripting languages have operators of comparisons to compare a lot of different things. Here is the syntax for the MATLAB scripting language operators of comparison:



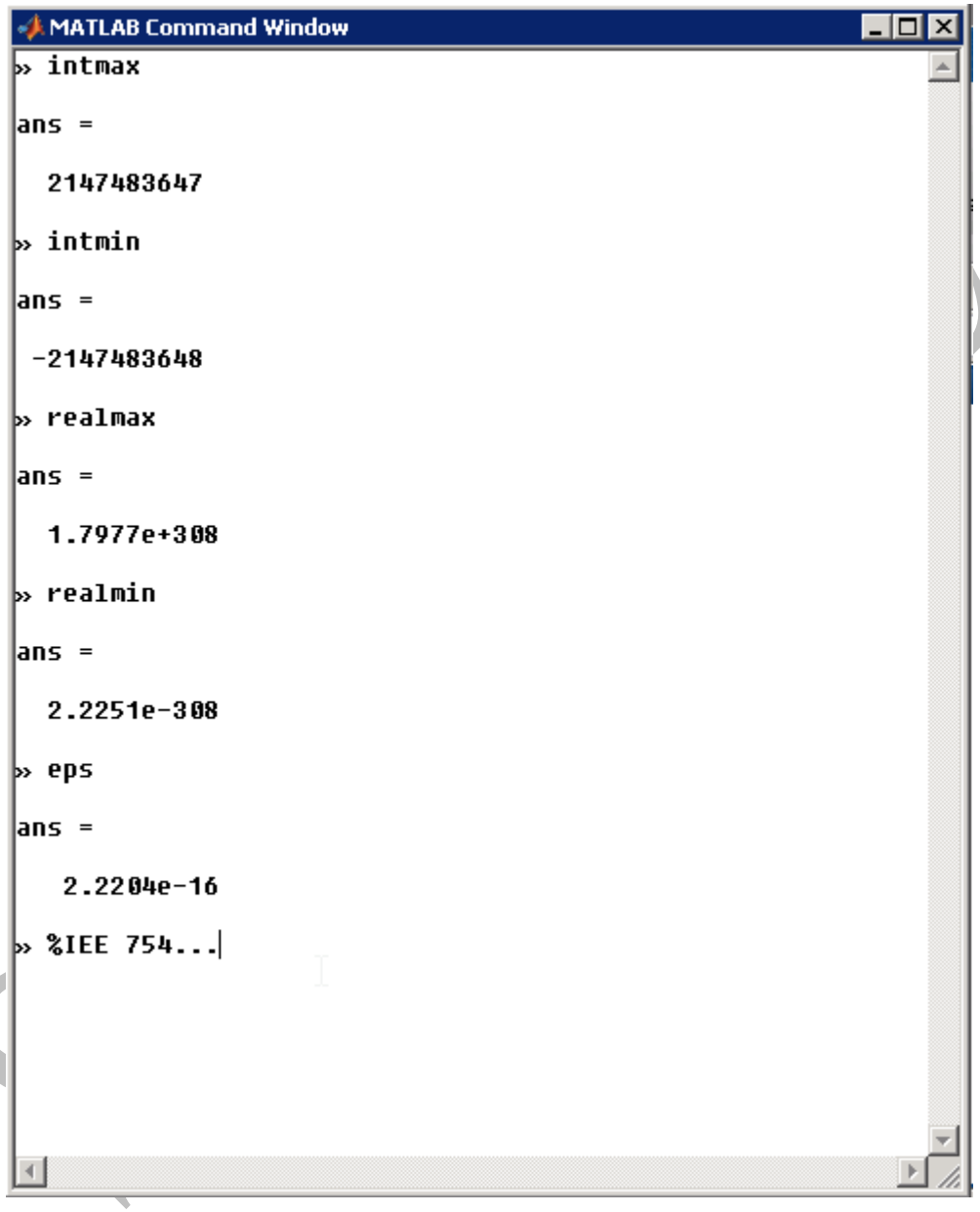
But you try to compare two texts you will get a surprise because in MATLAB everything is matrices (also vectors) in reality texts are also vectors as you can see it below:



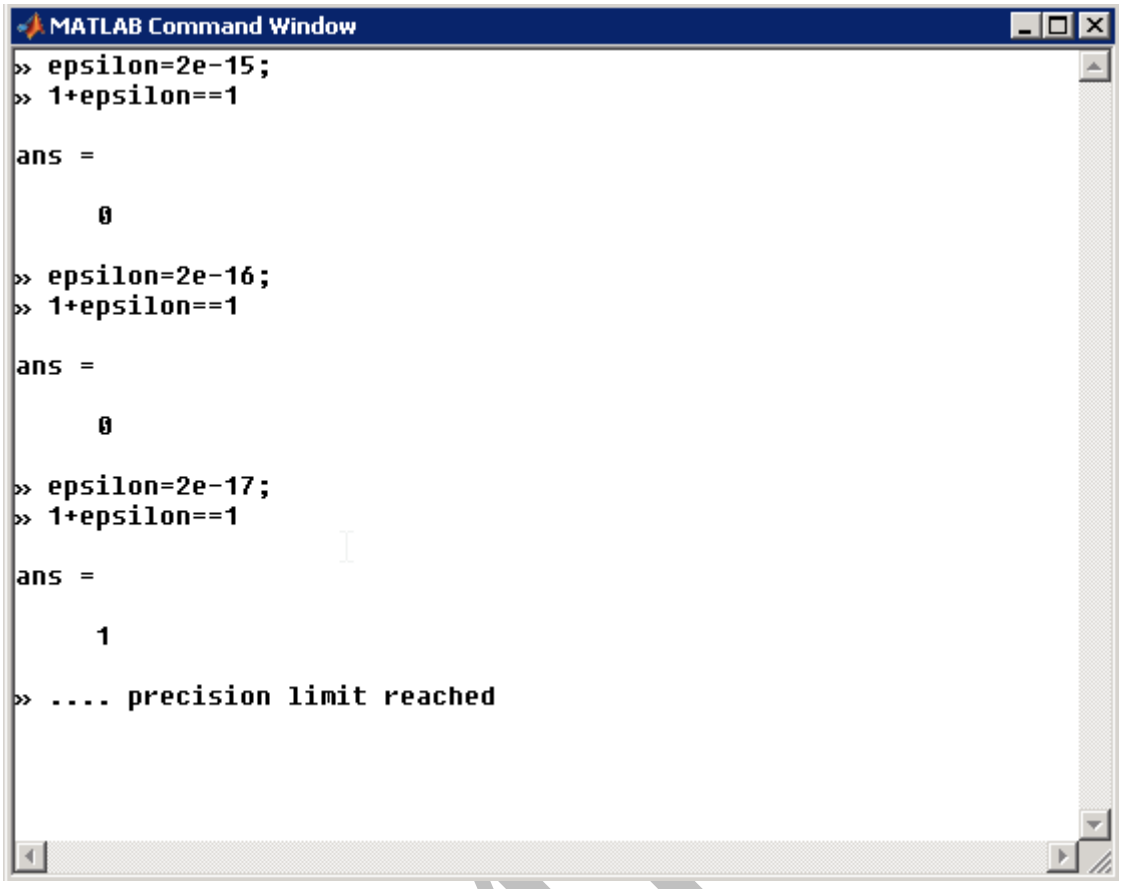
SAMPLE

Real numbers manipulations

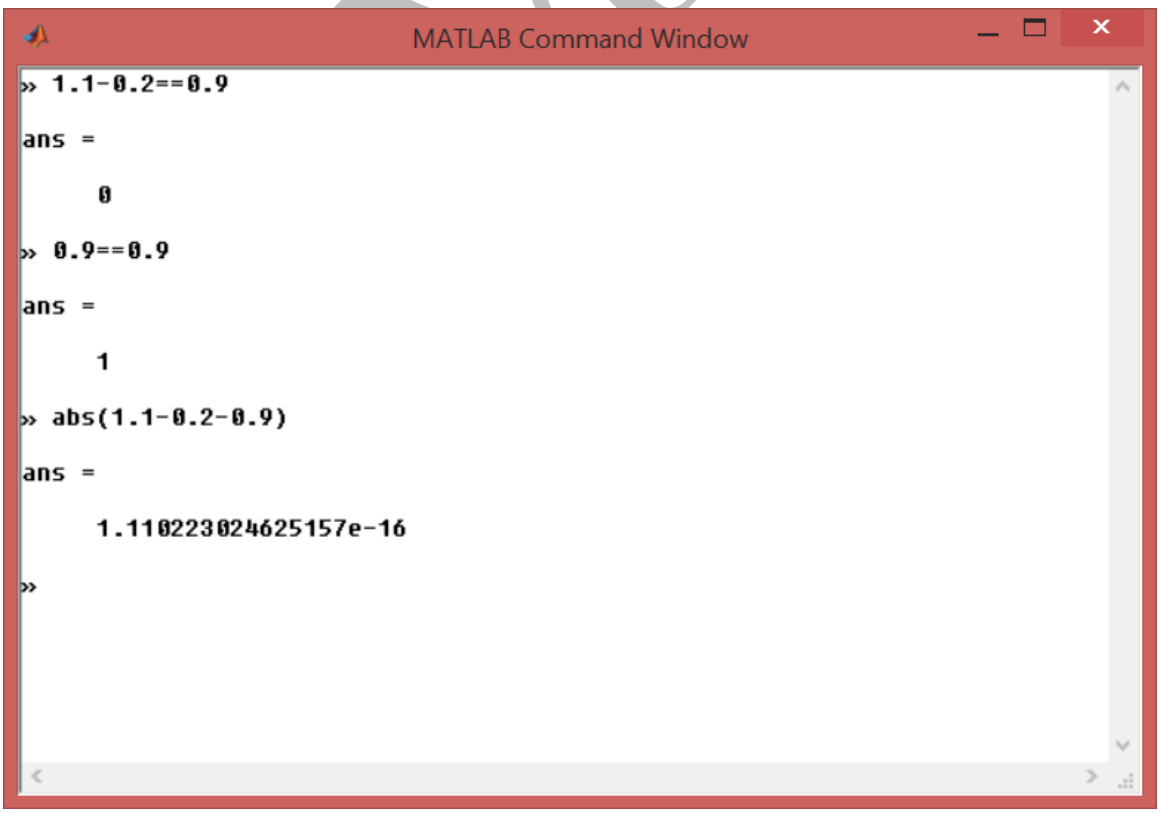
The first interesting thing when you work with any software is to know the maximum and minimum viewable values that the software can manage. You can get these information in MATLAB by using the commands **intmax**, **intmin**, **realmax**, **realmin**:



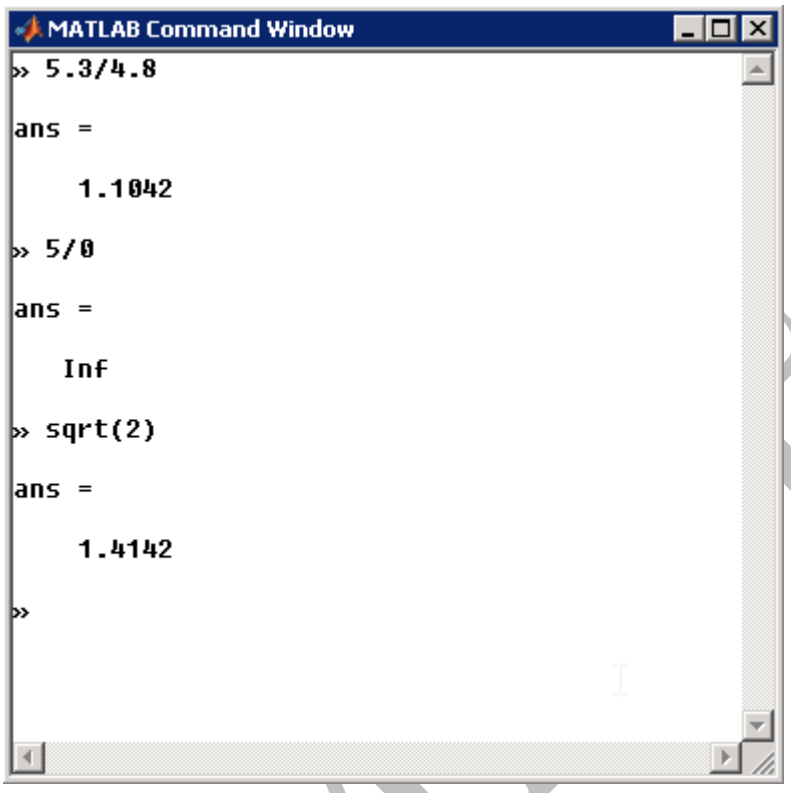
But take care! This does not means that MATLAB calculates with an accuracy of 2.2251e-308. In reality he is limited like most software to IEEE double precision as you can see it here below:



or by seeing this counterintuitive example:



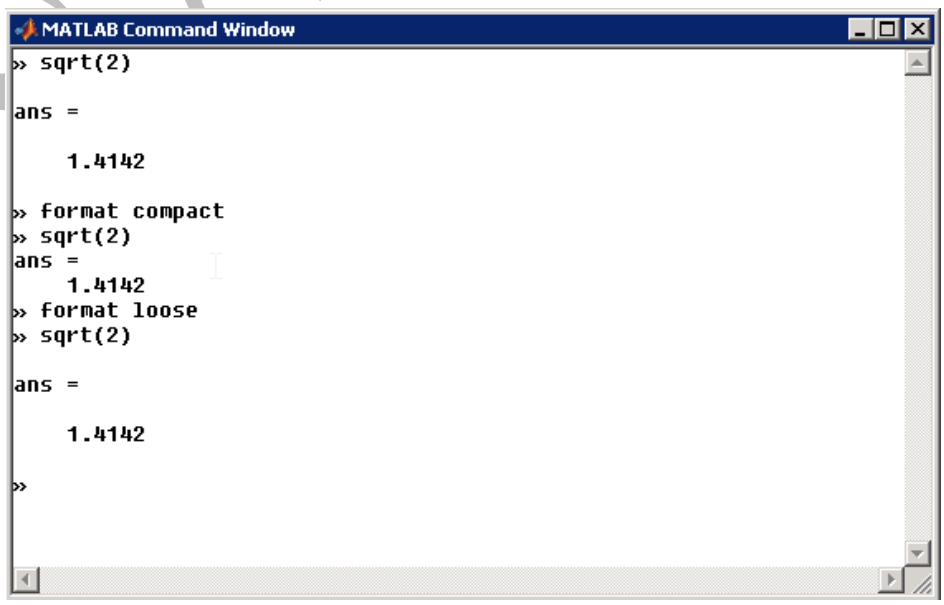
As you can see here below real numbers are otherwise correctly management by MATLAB and we have an important value (that we will use a lot in Optimization toolbox and Symbolic) for infinity that is **Inf** or **–Inf**:



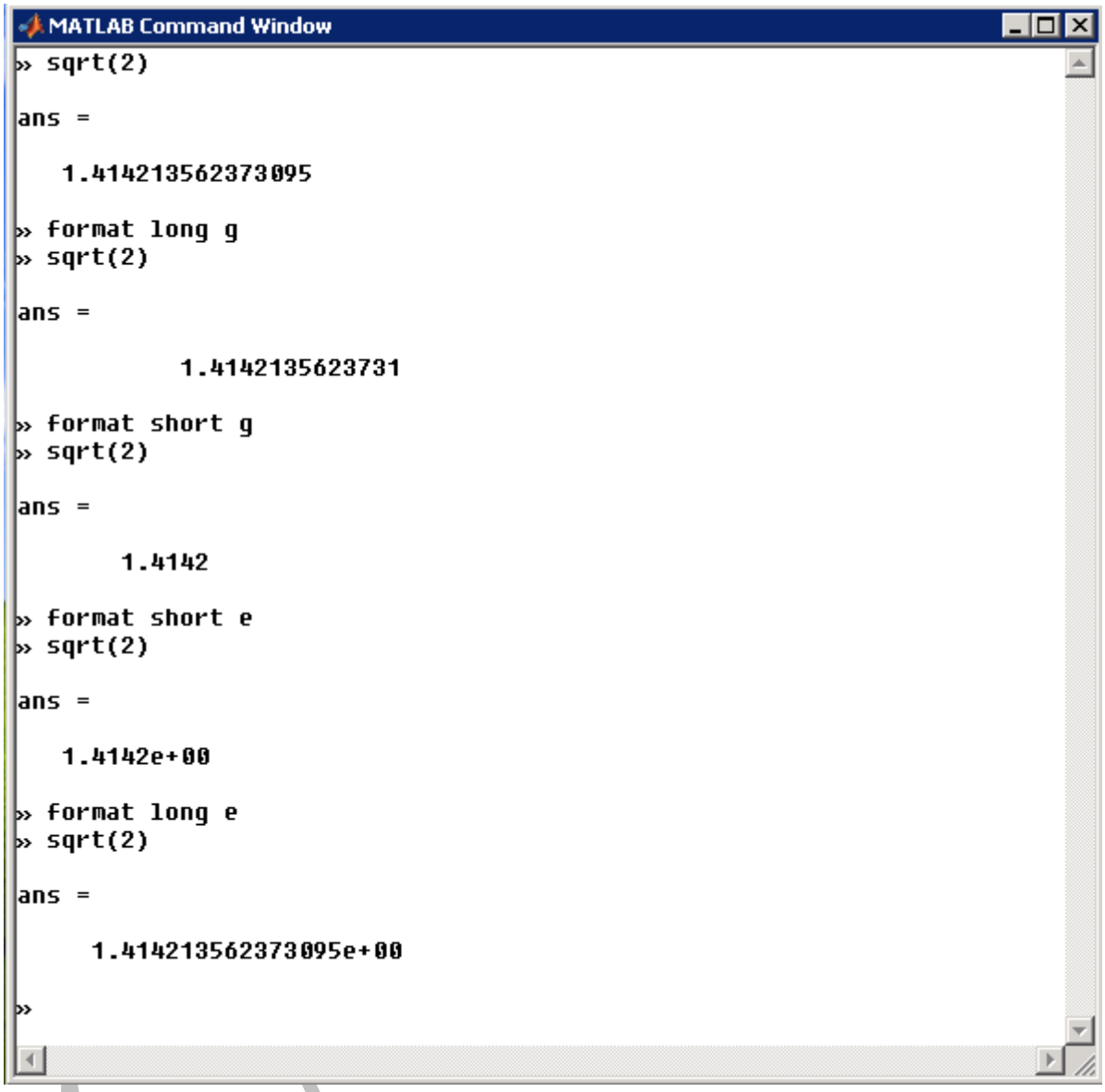
Command Windows real numbers formatting and output

When make calculations in MATLAB there is a boring white space between each line. This space is useful when you teach MATLAB but not when you are using it alone at your office.

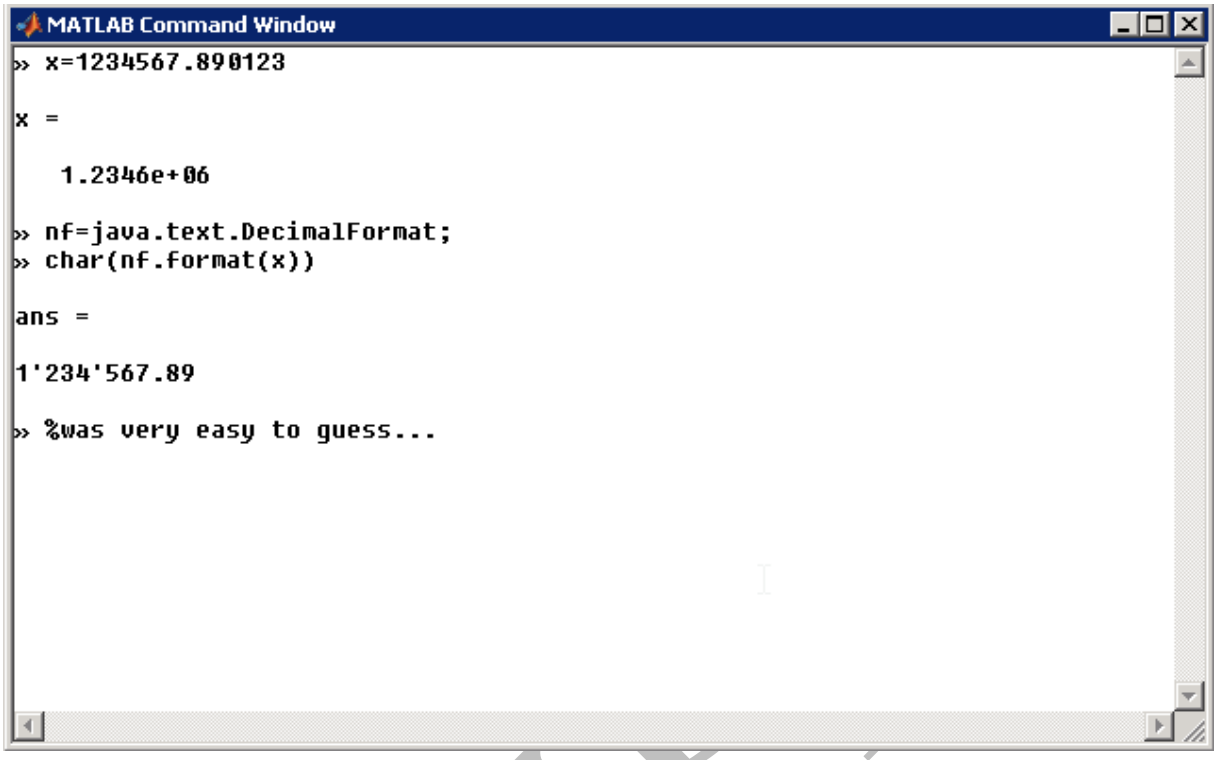
To remove the white space use the command **format compact**, to reset to the previous state use **format loose**:



Otherwise you can also choose the precision of number outputs using different commands like **format long g**, **format short g**, **format long e**, **format short e**:



And what about thousands separators?:

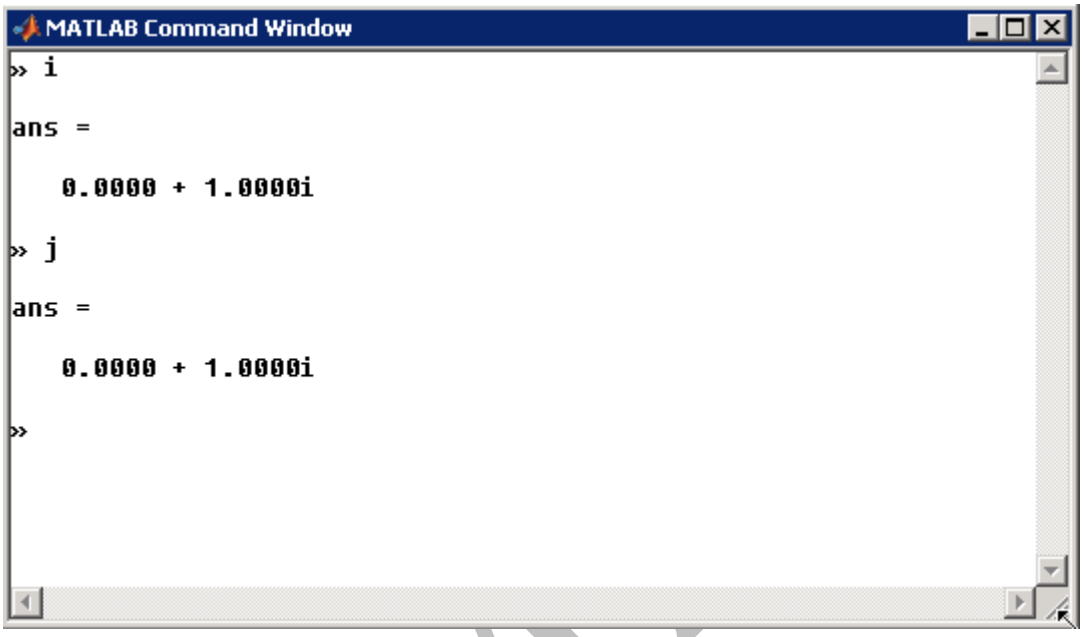


SAMPLE

Arithmetic operations on complex numbers

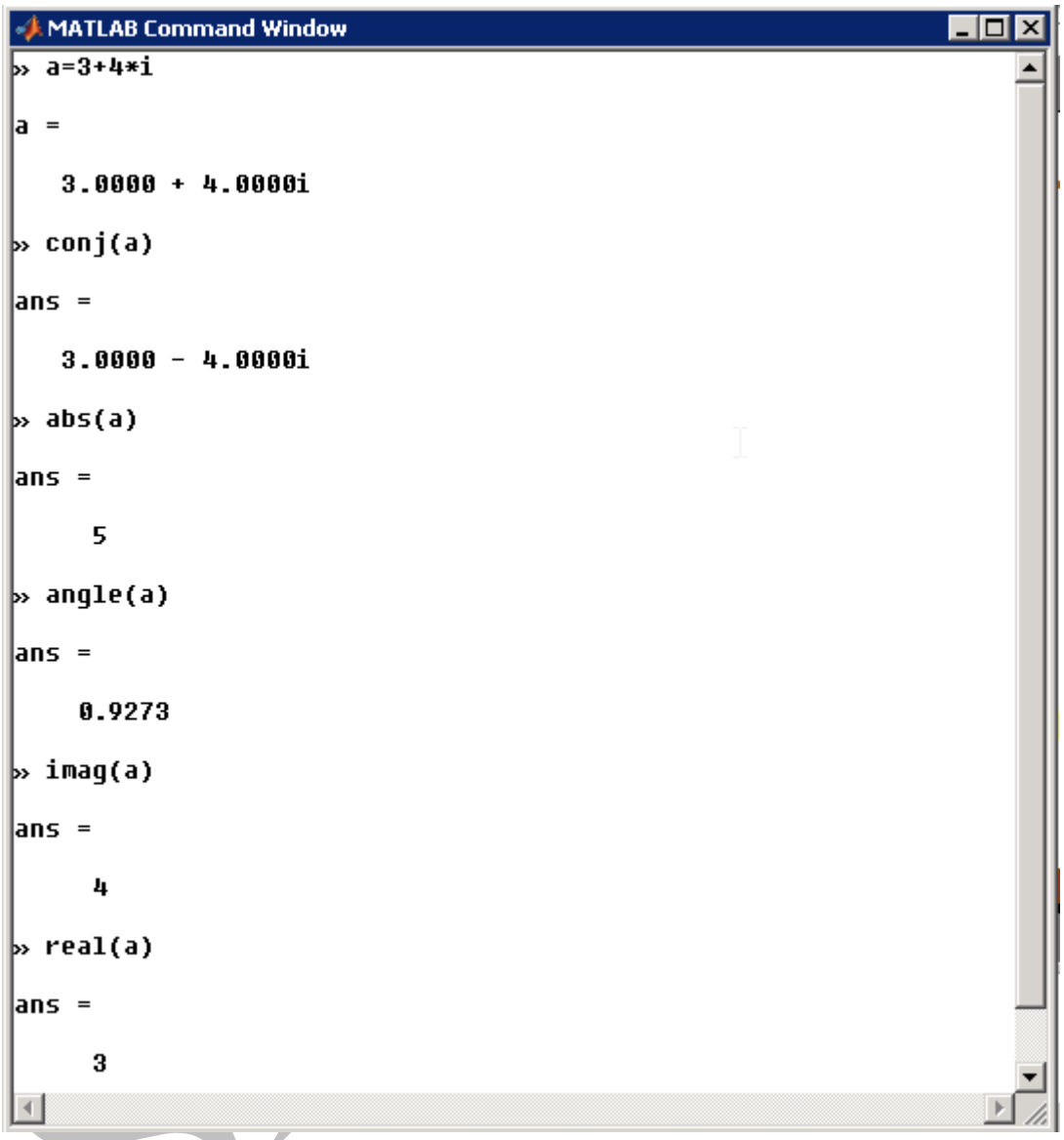
Complex numbers are especially useful in electronics/electordynamics and physics but you will also have to manage them for example in Finance!

First you must know that MATLAB can use the letters **i** or **j** as complex number imaginary part.



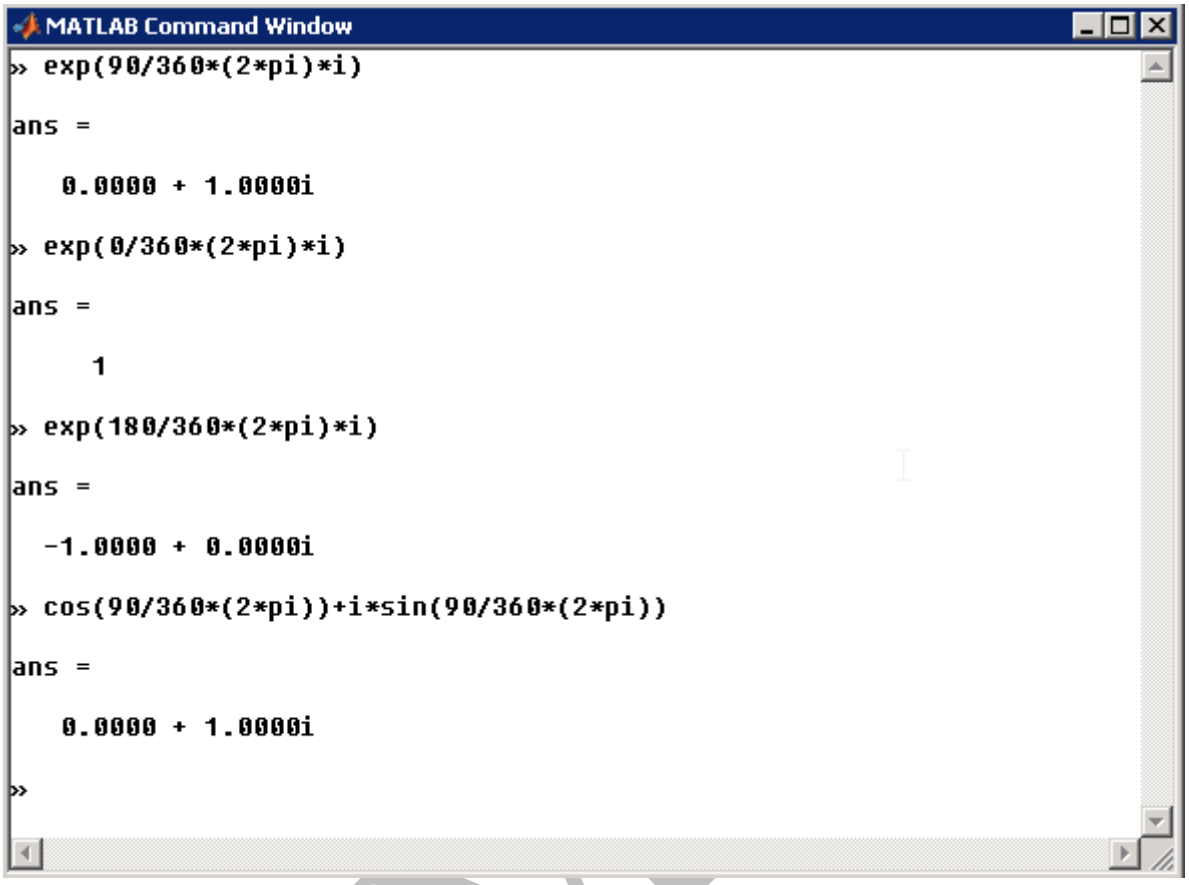
SAMP

You can otherwise make some very classic calculations on a complex number like addition, multiplication, division, power, root but also get the conjugate with the function **conj( )**, the module with the function **abs( )**, the angle of the phasor with the function **angle( )**, and the imaginary and real part with the functions **imag( )** and **real( )**:



And for just the classical +,-,/,*,^ operators can also be applied on complex numbers.

For some charts (especially the **compass**!) it is useful to remember the different technics to get a complex number just from an angle given degrees:

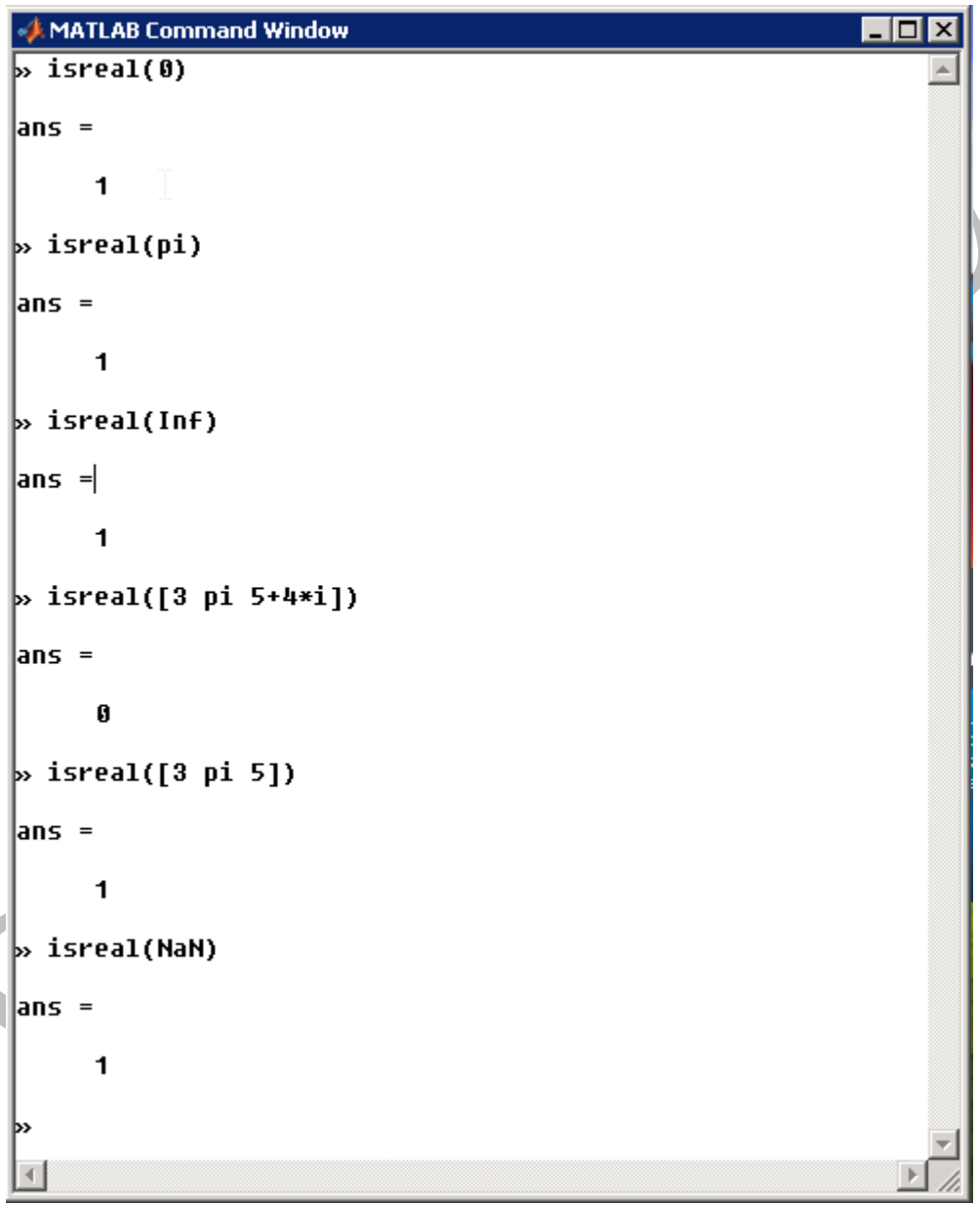


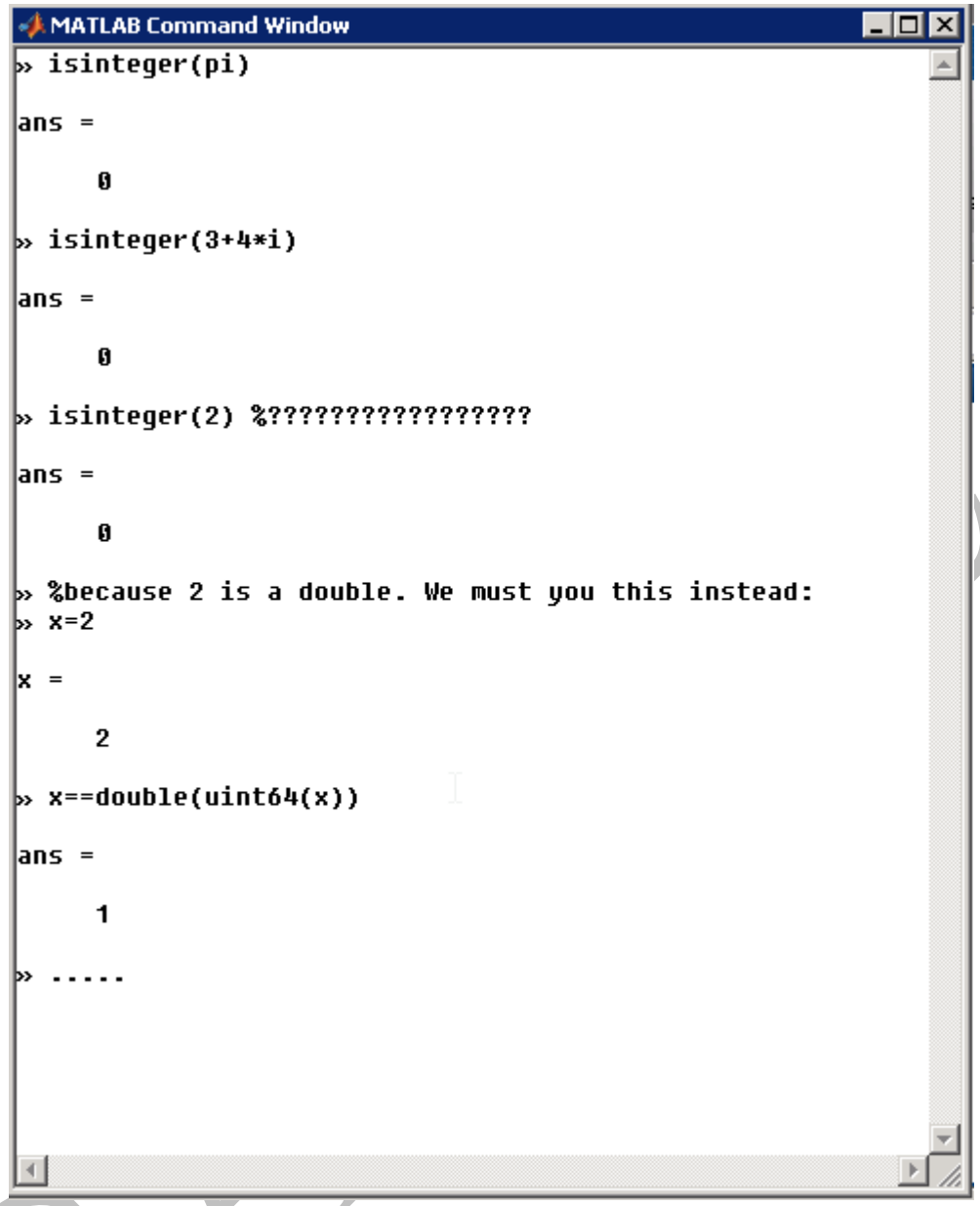
SAM

Identify the number type

Many times you will have to face with functions that will return multiple solutions. It is then necessary to know how to determine if a result is a real, integer or complex.

For this purpose you can use the functions **isreal( )** or **isinteger( )** on scalar or vectors as shown below (**iscomplex( )** does not exist!):

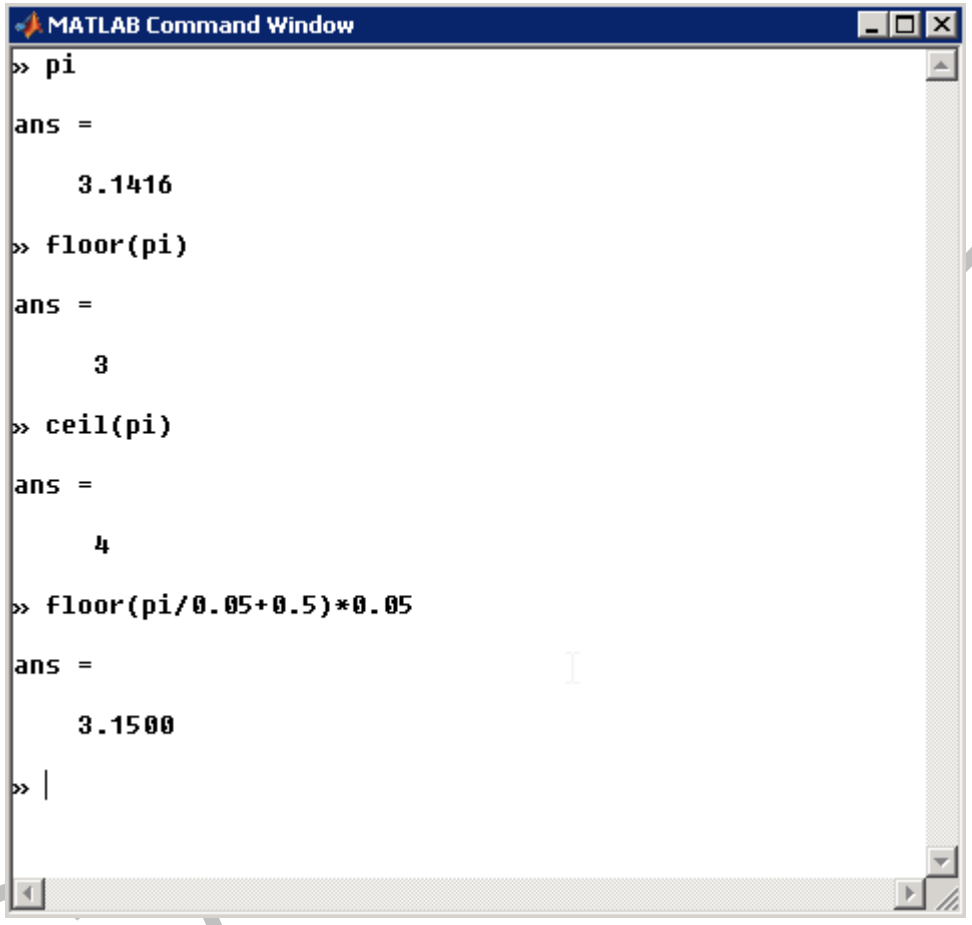



MATLAB Command Window
» isinteger(pi)
ans =
0
» isinteger(3+4*i)
ans =
0
» isinteger(2) %???????????????
ans =
0
» %because 2 is a double. We must you this instead:
» x=2
x =
2
» x==double(uint64(x))
ans =
1
» .....

Rounding results

Rounding results is always something important if you want to generate a nice output on a chart or in an automated report.

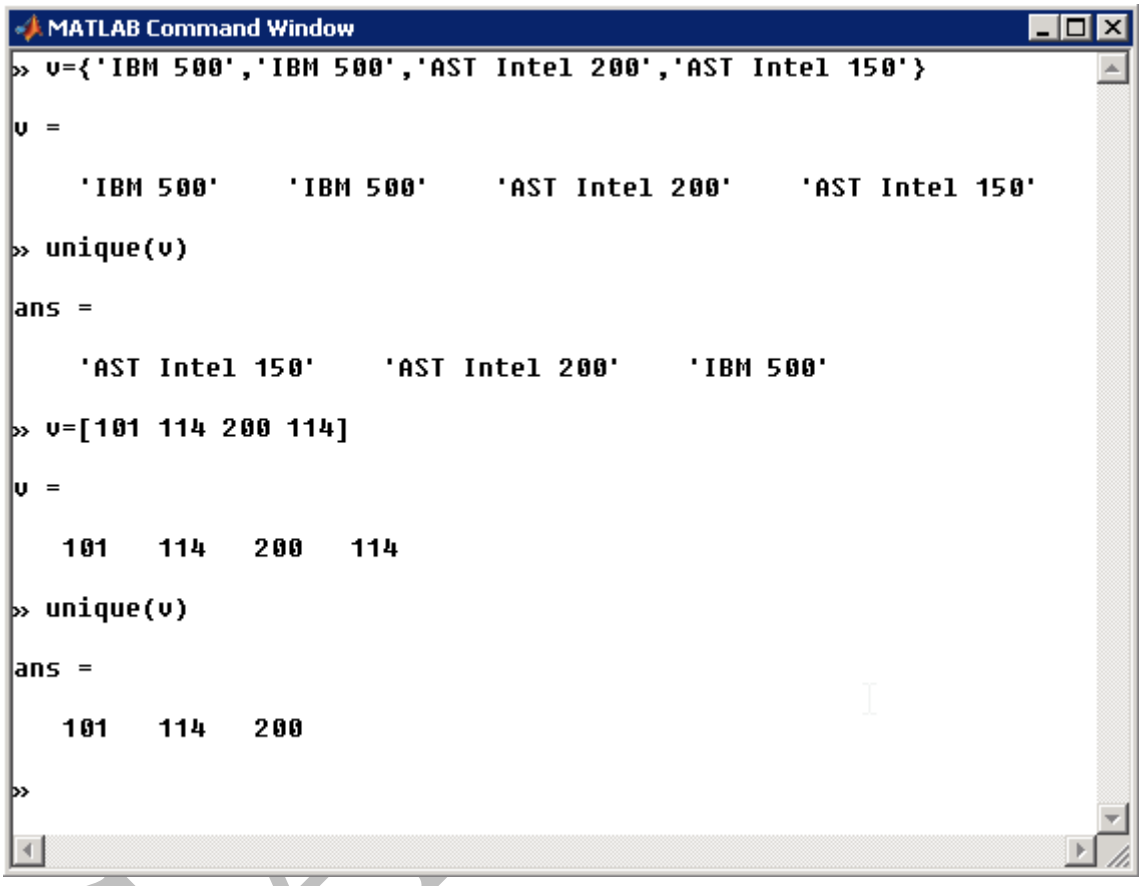
For this purpose MATLAB has two important functions **ceil( )** and **floor( )** that can also be used with the famous equation to round to any multiple we want (in the last example below we round to the nearest 0.05):



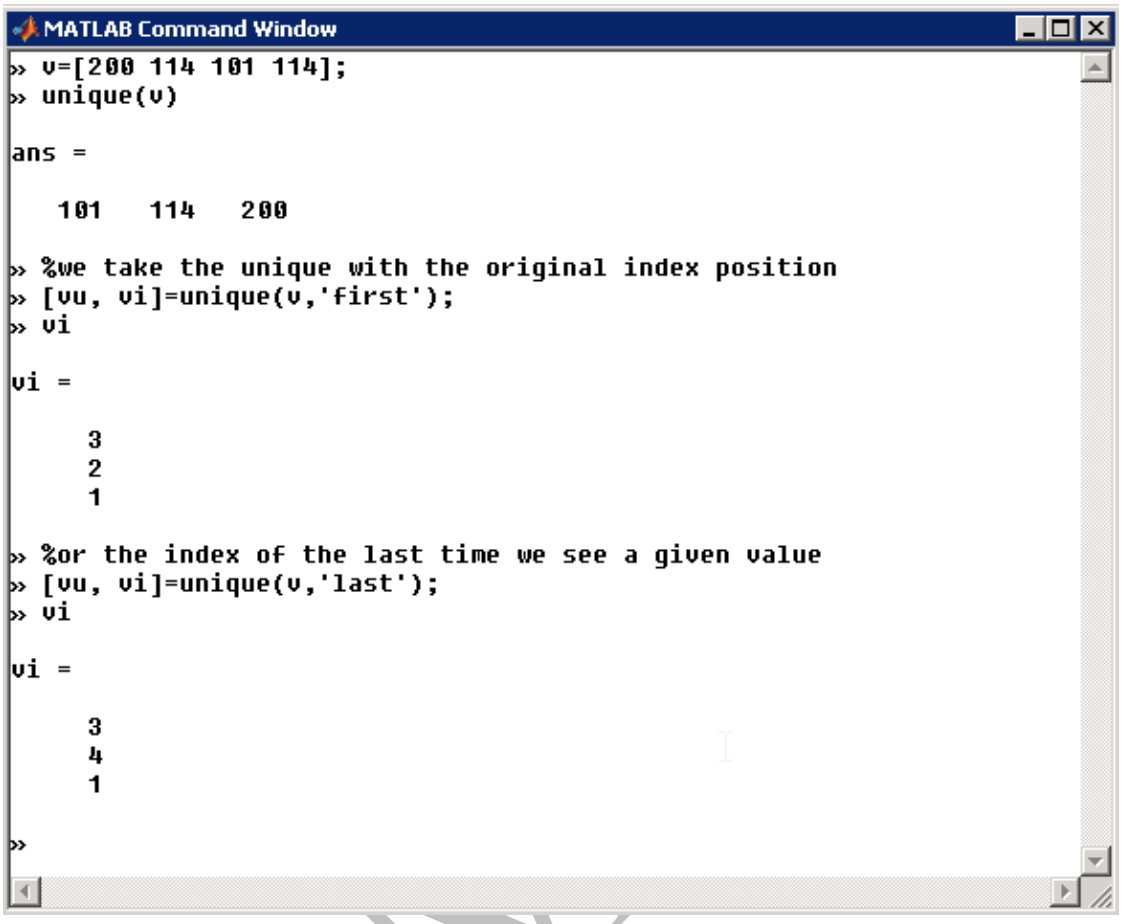
Set (vectors) operations

Numerical vectors and string vectors (also name "cell arrays" in MATLAB") are one of the most important objects that any user has to master. Before making calculations with such objects one important prior knowledge to have is to be able to apply set operations on them.

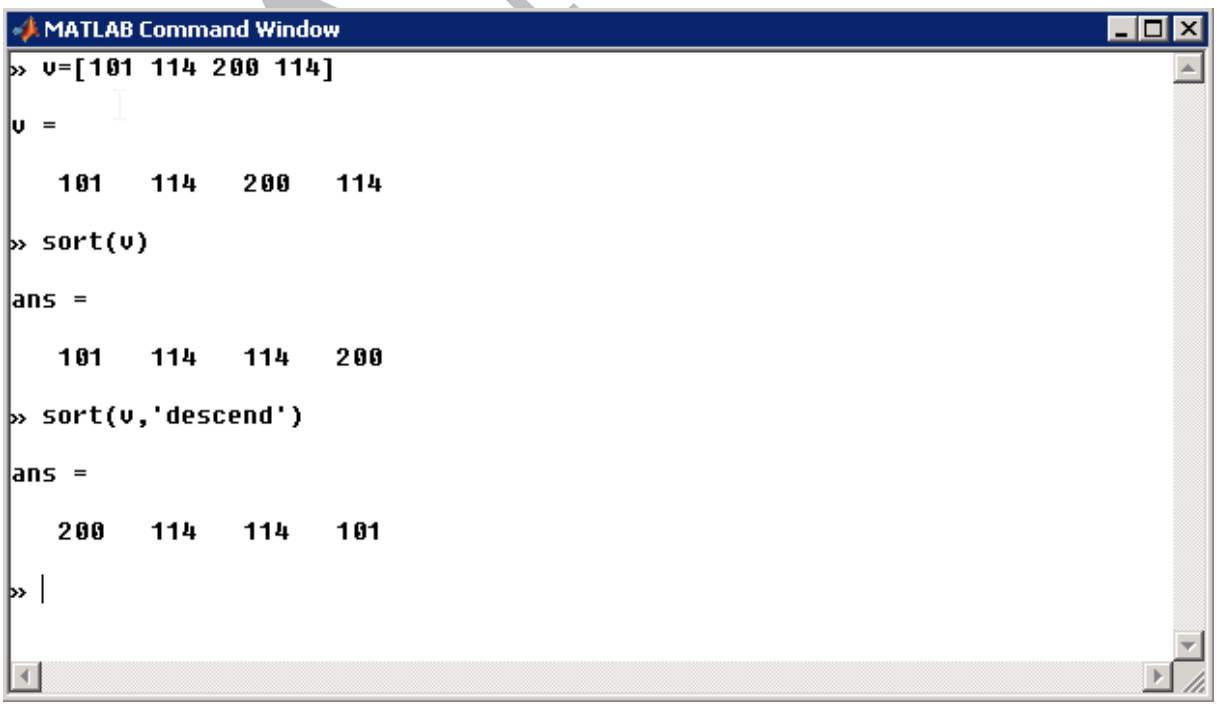
First see how to define a simple string vector (cell array) and a numerical (line) vector and how to extract unique items using the **unique( )** function:



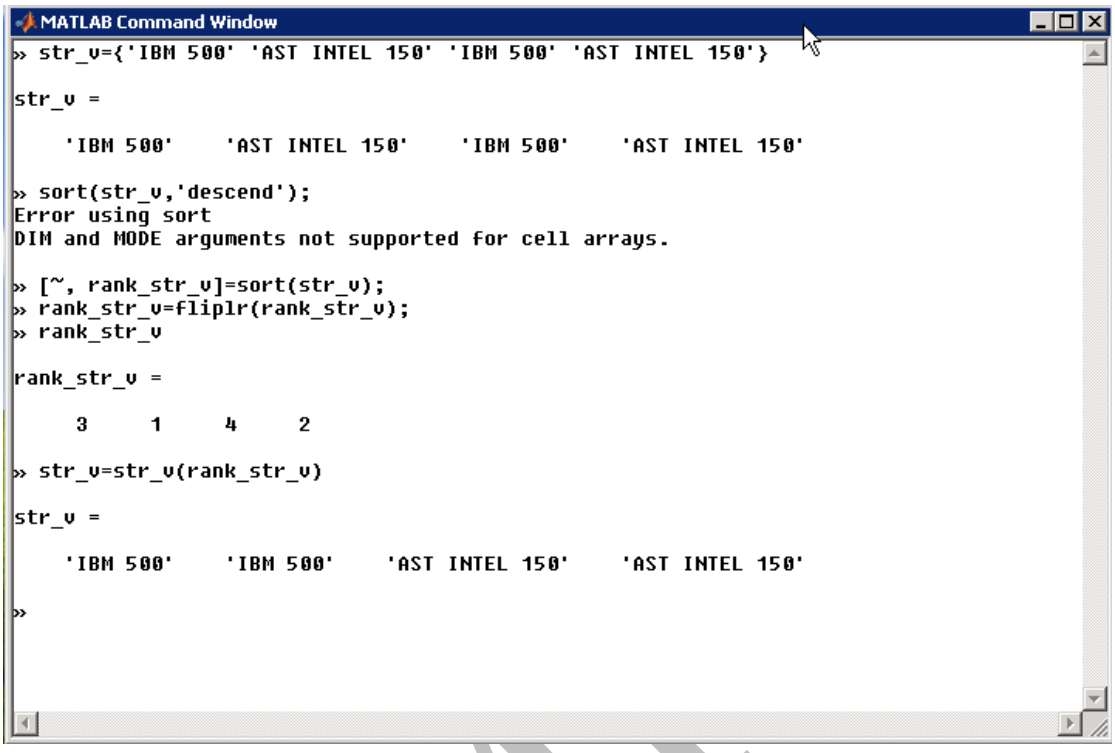
But in facts, the function **unique( )** is one of this numerous MATLAB functions that returns multiple output. For example we can see below hot to get also the ranks:



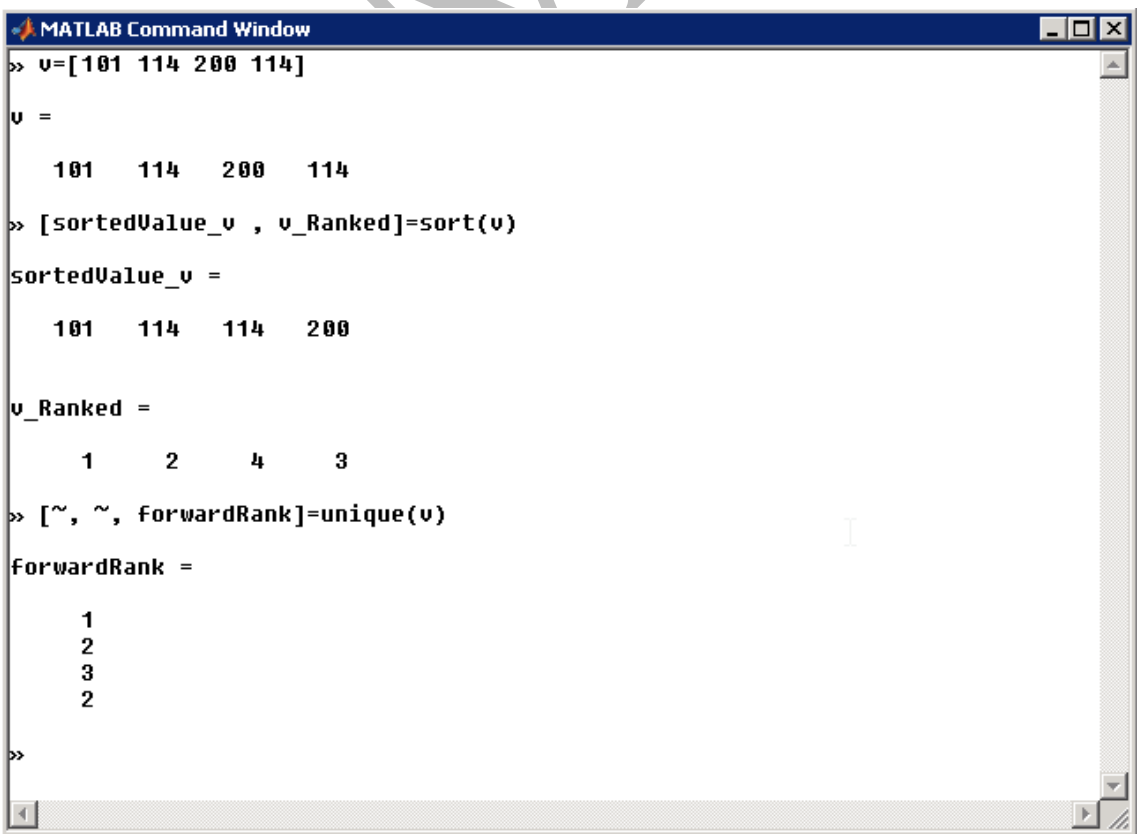
We can also sort in a very easy way numerical vector using the **sort( )** function:



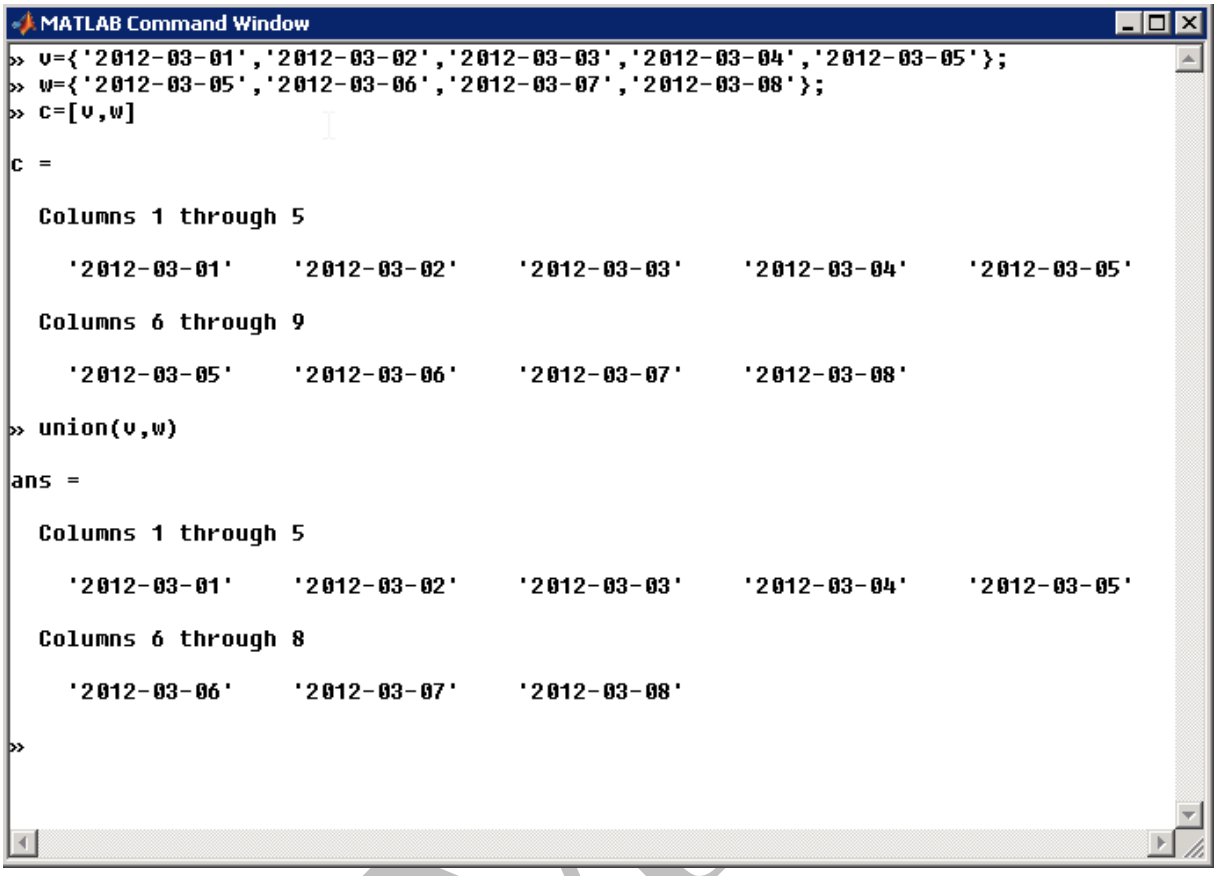
But sorting string vectors (cell array) **in a descending order** is a little bit more complicated. We have to play with multiples functions using **sort( )** and **fliplr( )** together to inverse the sorting order of the ranks and then use it to get the final string vector:



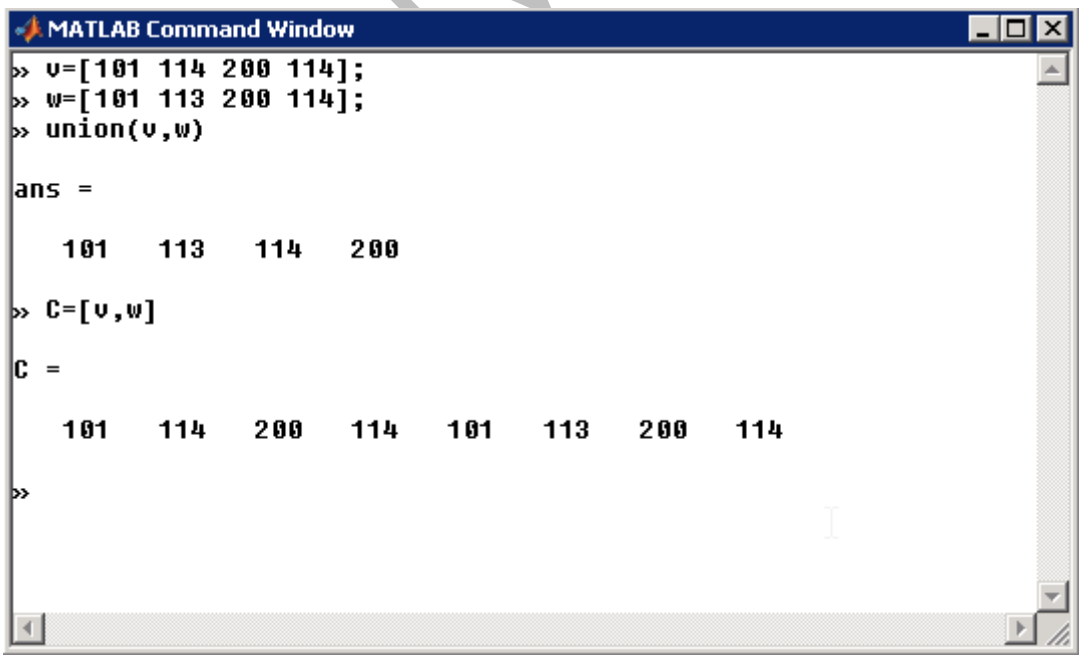
An interne thing in rankings is to manage equalities. See how MATLAB can manage that with the function **unique( )**:



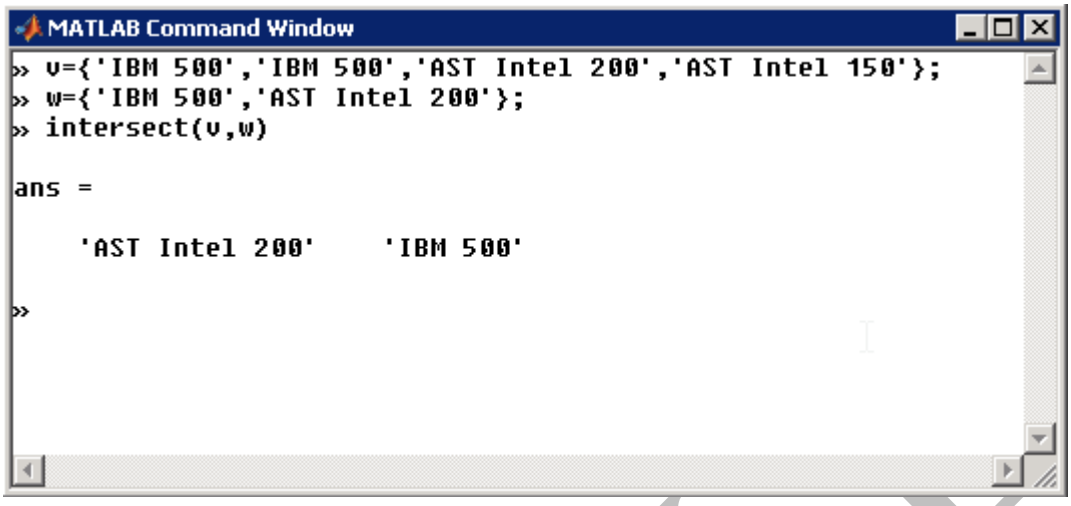
Something also important with vectors (or cell array) is to learn to merge them or to make a union set operation using the function **union( )** to avoid duplicates:



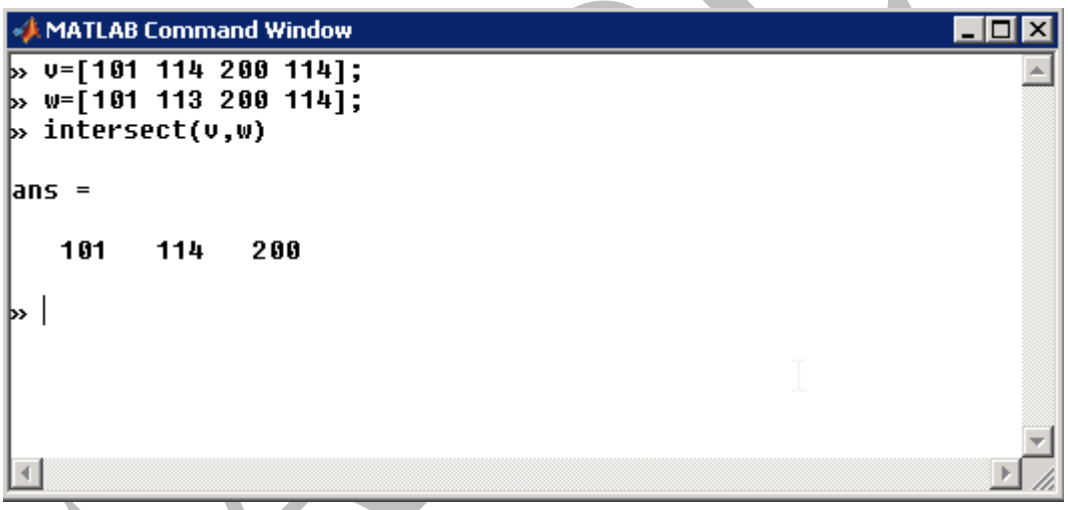
That works also the same for numeric vectors:



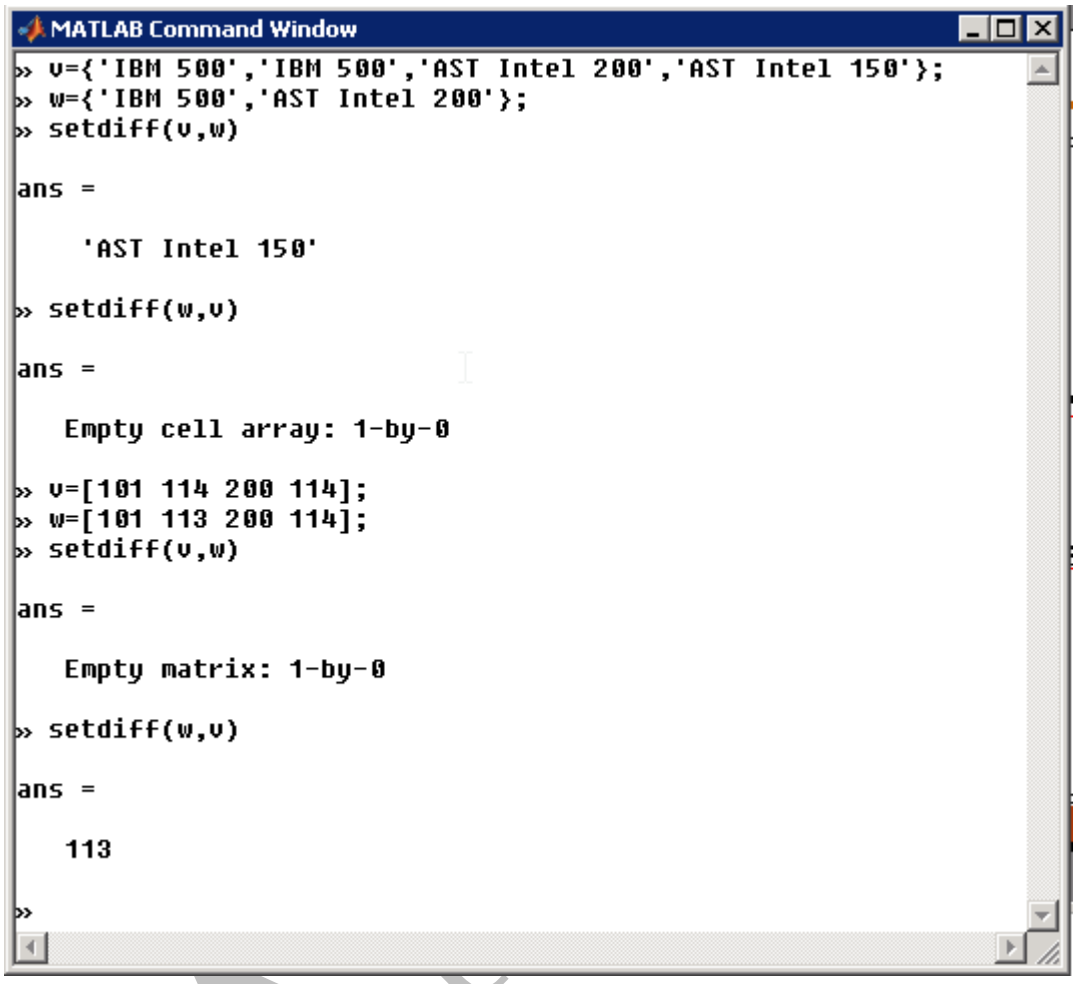
If you are interested to get the similarities between a given number of vector (numeric or strings) you can use the set operation function **intersect( )** as shown below:



And here we see that it works the same for numerical vectors:

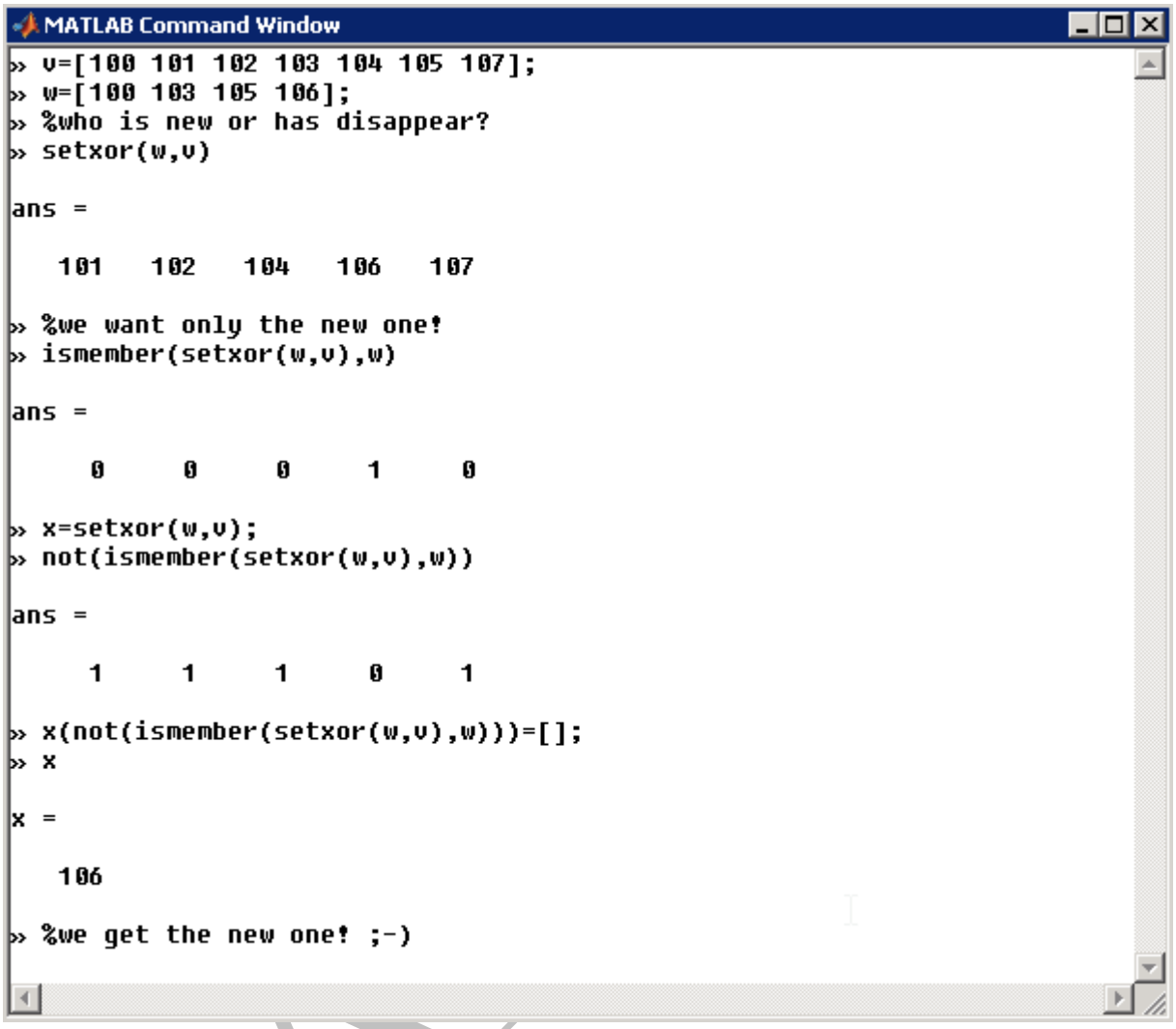


When we have two vectors (numeric or string) we can be interested to know what differs between them we can use the **setdiff( )** set function. **Don't forget that it is not a commutative operator!**:



SAM

We can also use the **setxor( )** set operator function to investigate what has change between two vectors (numeric or string) as you can see below:



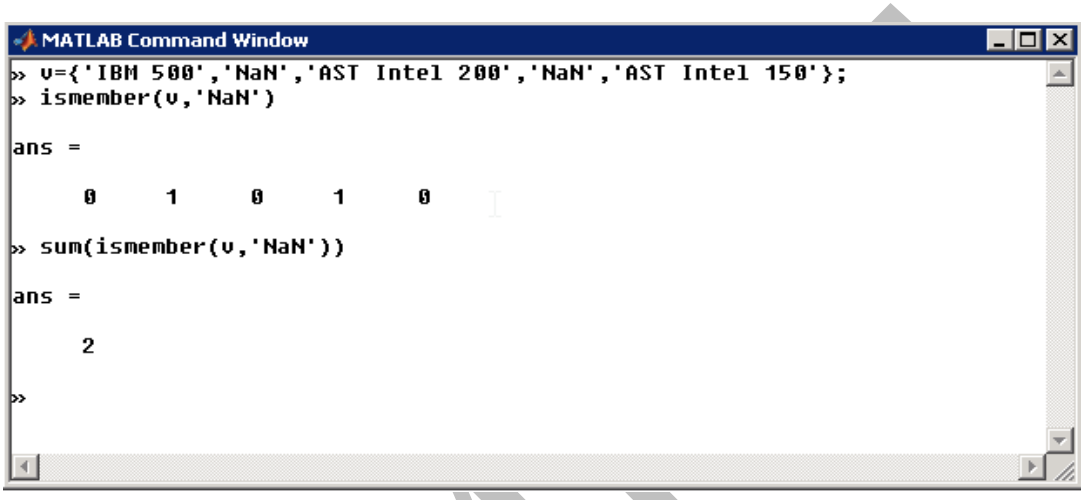
Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seems to have built-in functions.

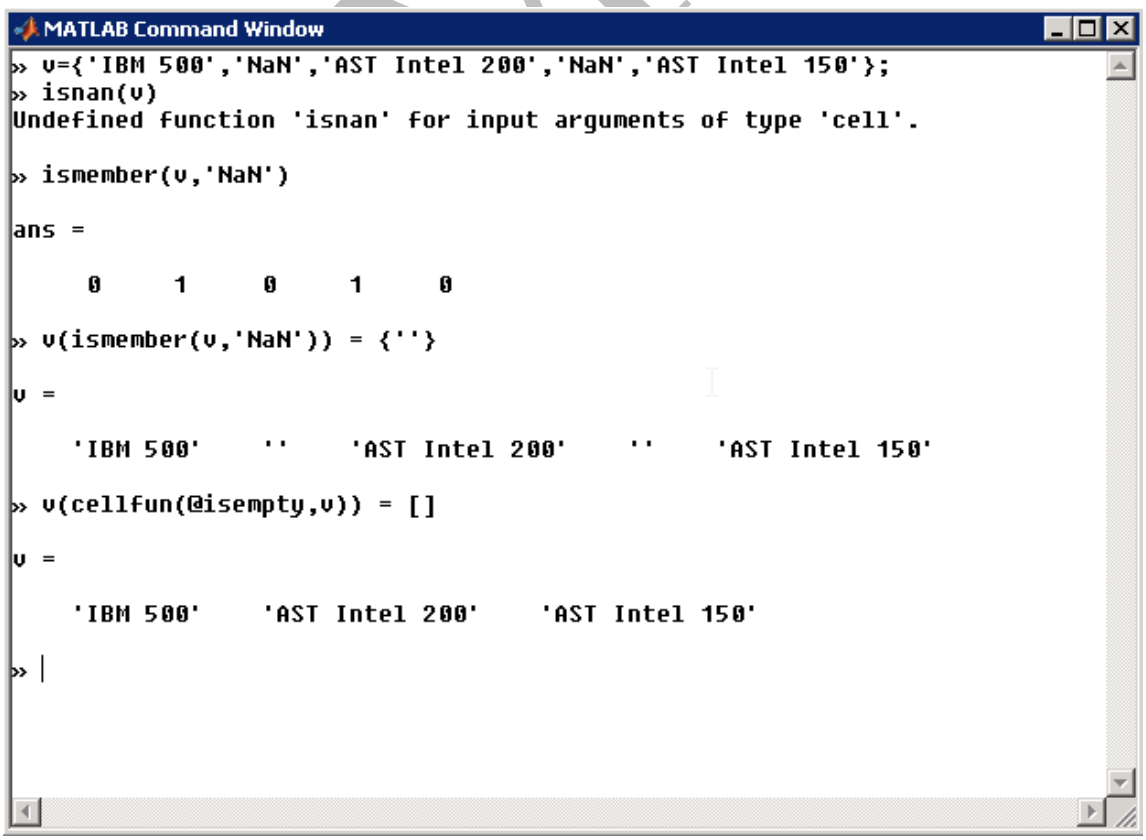
Counting/Removing NaN in cell arrays

The first typical question is to remove **NaN** values in a cell array.

How many NaN do we have? Use for this the **ismember( )** and **sum( )** functions together:

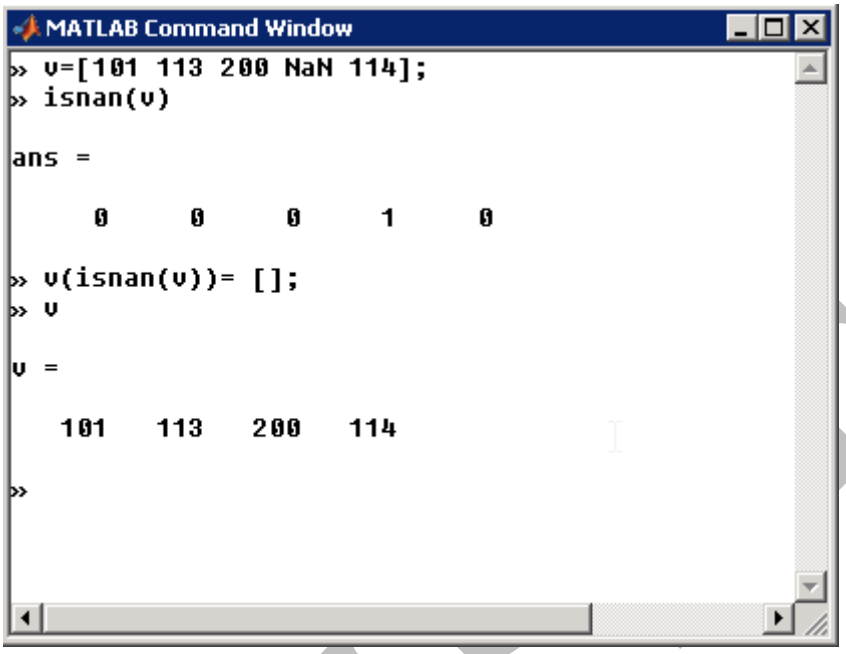


And now comes the tricky part... how to clean the vector using **cellfun( )** function:

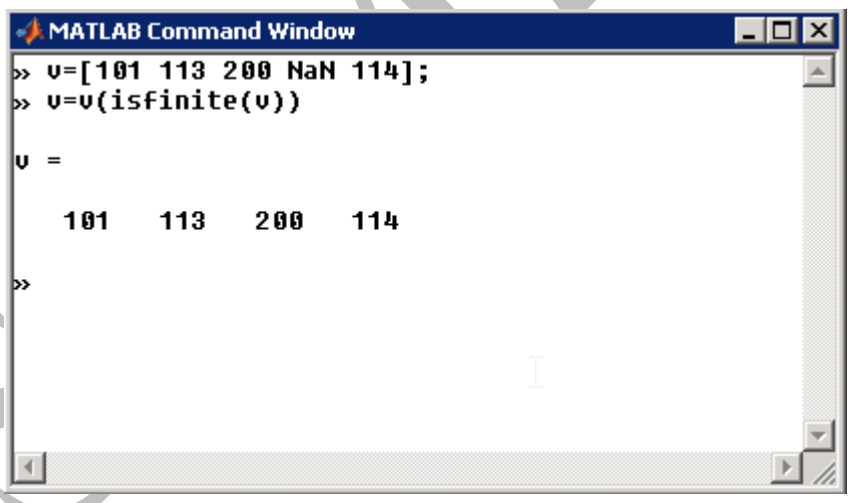


Counting/Removing/Replacing NaN in (numeric) vectors

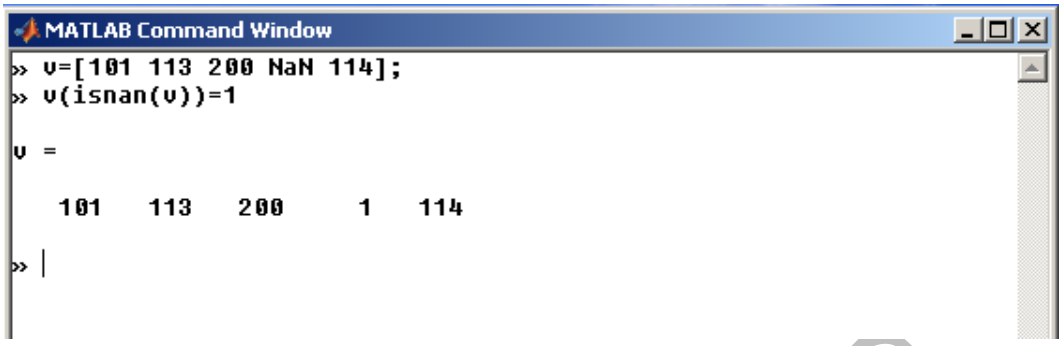
There are multiple ways to remove NaN in numeric vectors. There are all easier than with cell array. We will just use **isnan( )** function:



Or **isfinite( )**:



Now let us see how to replace NaN values what is very useful in finance or statistics:



Another interesting special case is the following:

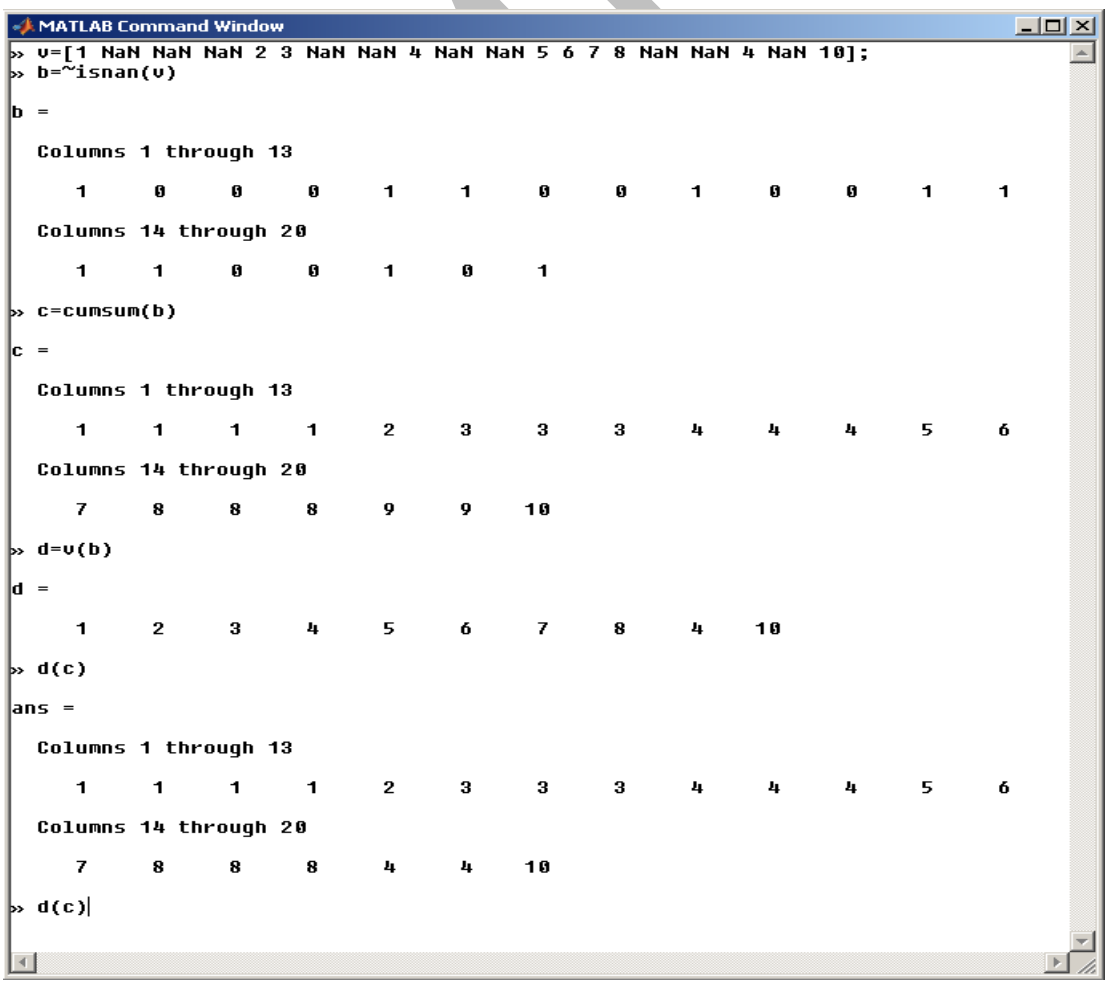
We want to replace NaN values in a large vector such as the one below:

v = [1 NaN NaN NaN 2 3 NaN NaN 4 NaN NaN 5 6 7 8 NaN NaN 4 NaN 10];

and we would like the output to be

v = [1 1 1 1 2 3 3 3 4 4 4 5 6 7 8 8 8 4 4 10];

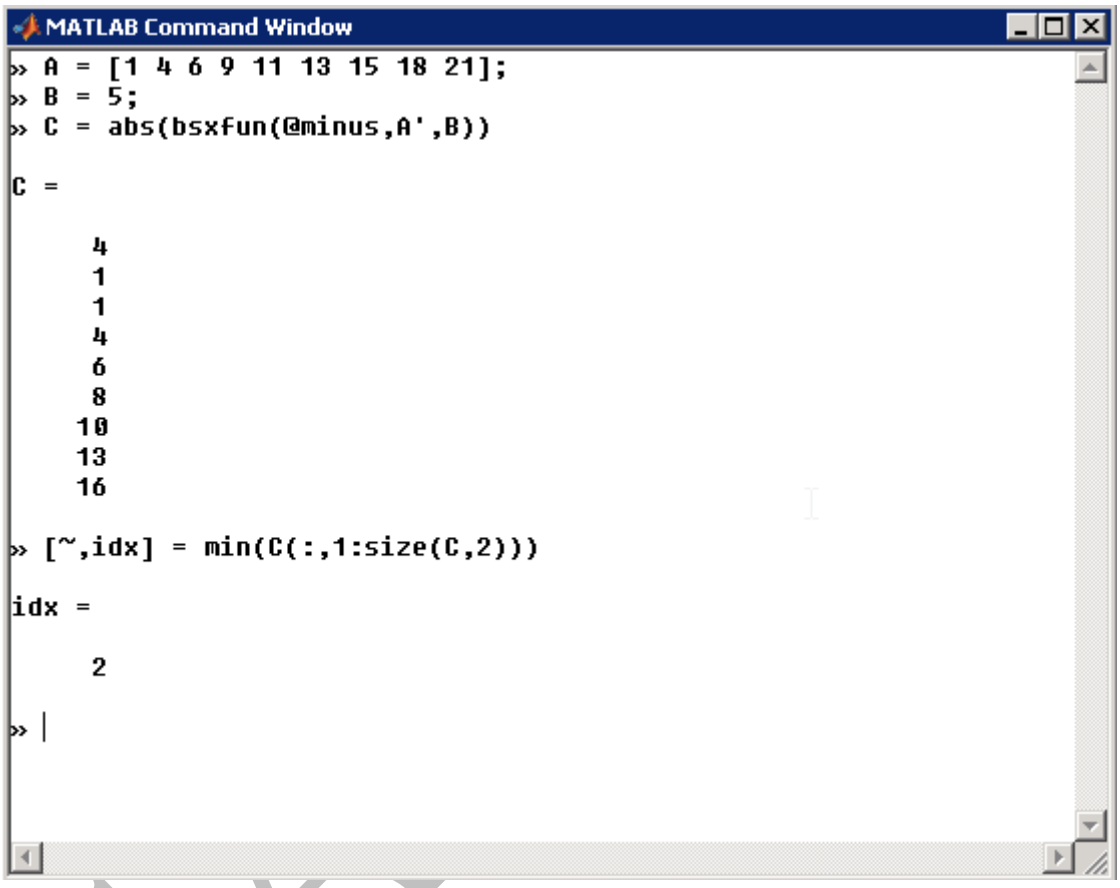
The solution is:



Finding the closest number in an vector

Here we will see how to find the position of the closes value in a numeric vector. This is very useful in finance. For this purpose we will use a trick well known in a lot of programming languages.

The presence of the function **bsxfun( )** is just to make the thing faster (see page 639 on fast scripting tricks):

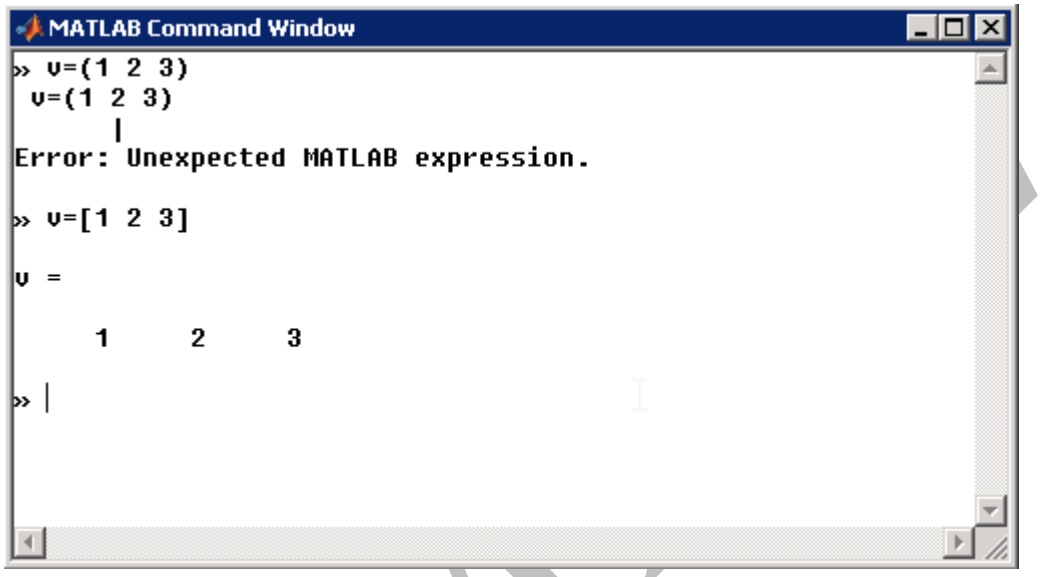


Arithmetic operations and manipulations of vectors

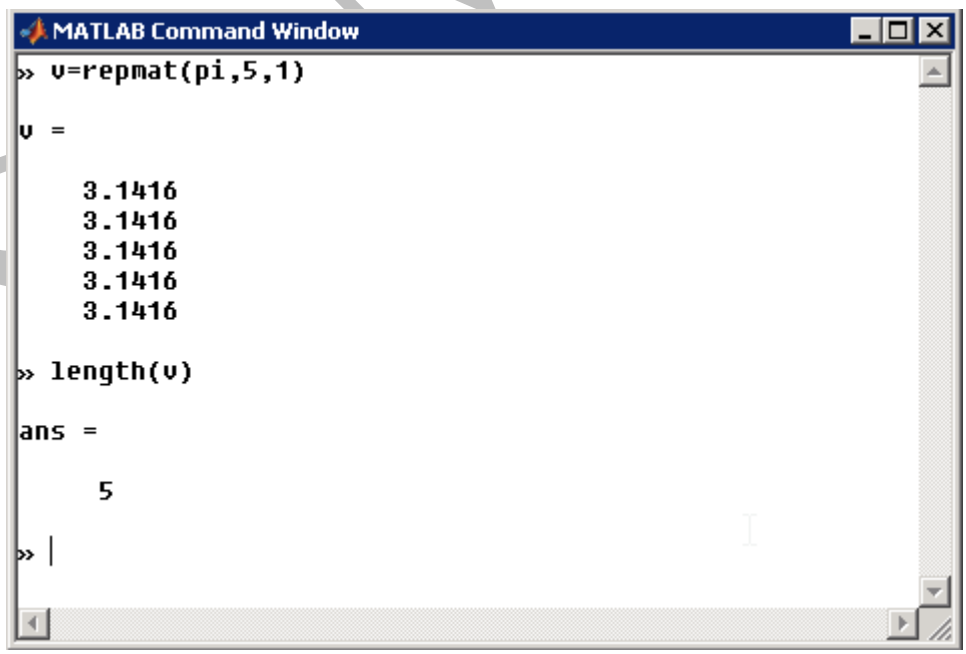
As we said before numerical vectors are one of the most important objects that any user has to master.

Let's see here the most common prior knowledge you must have before doing more complicated stuff (algorithms).

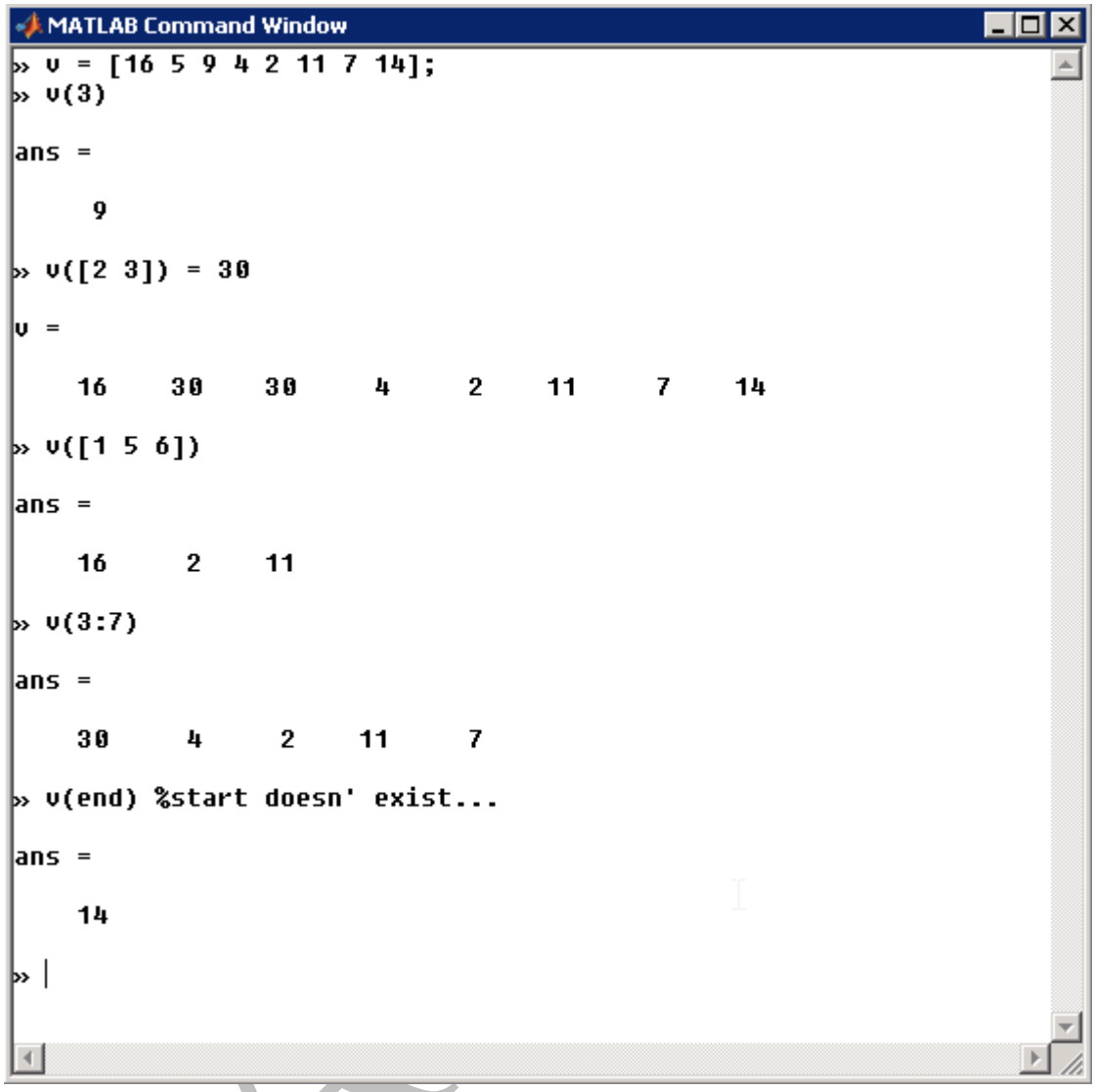
First you must remember that a vector will not be defined with parenthesis but with brackets:



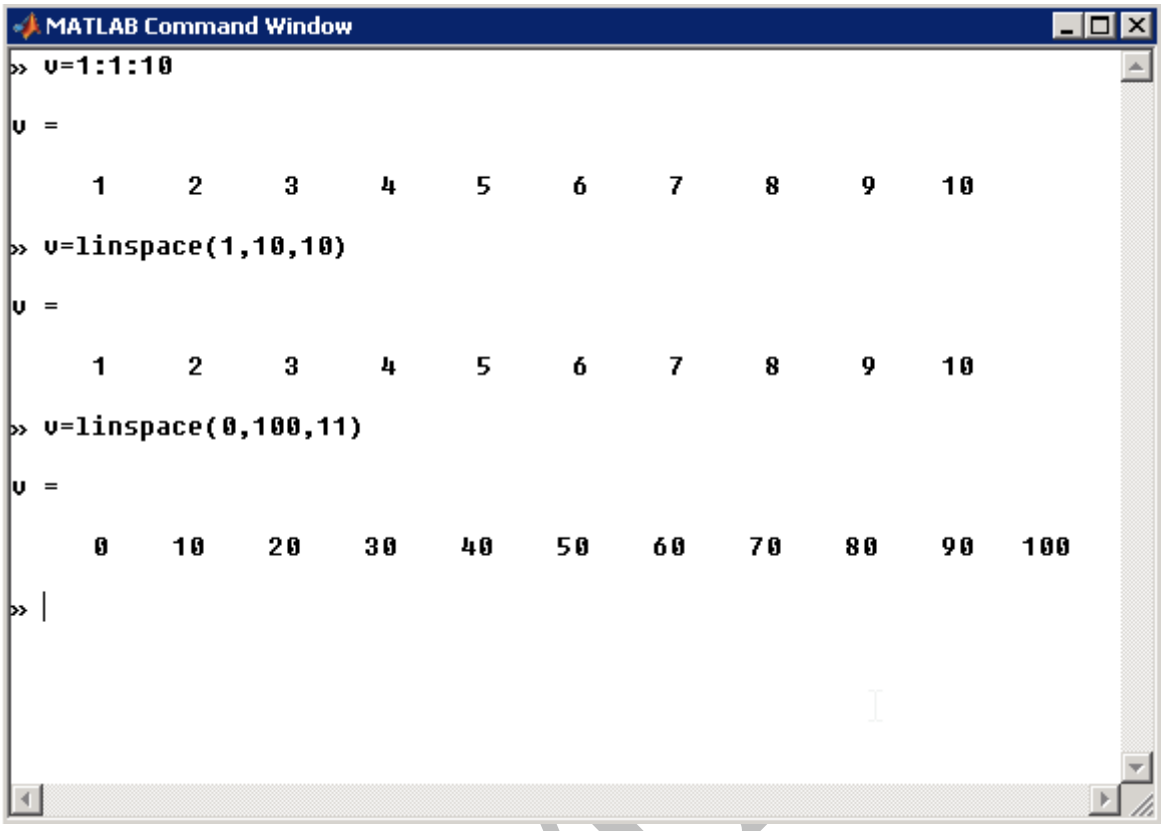
Another very useful function is to repeat a given value some times in a vector using **repmat( )** and also to get a vector **length( )**:



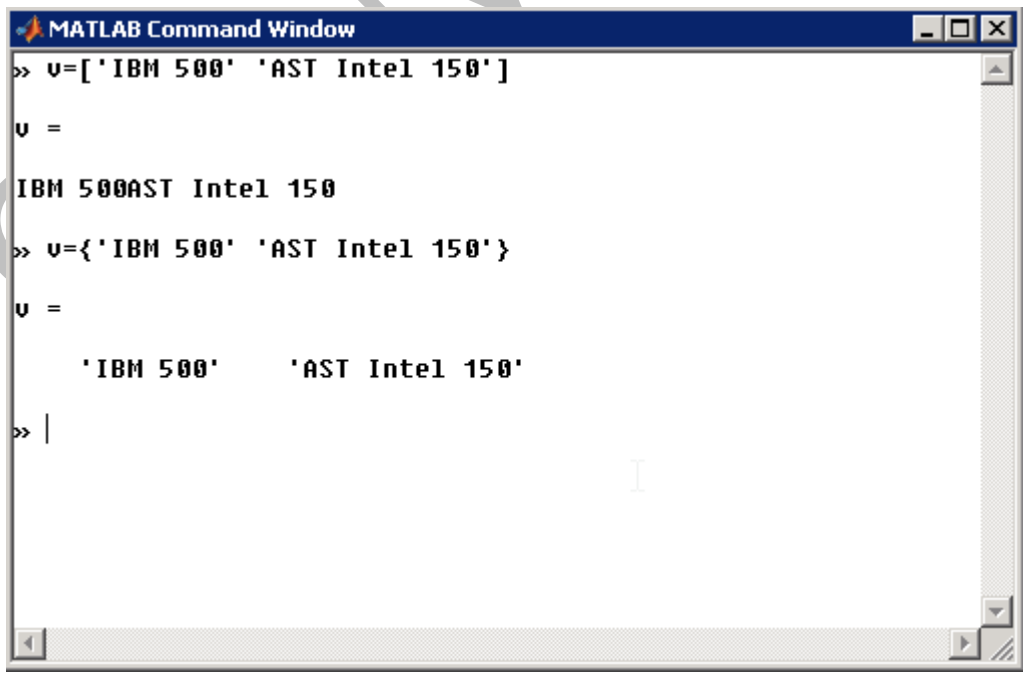
Extracting values from a vector or subsets is also something very important:



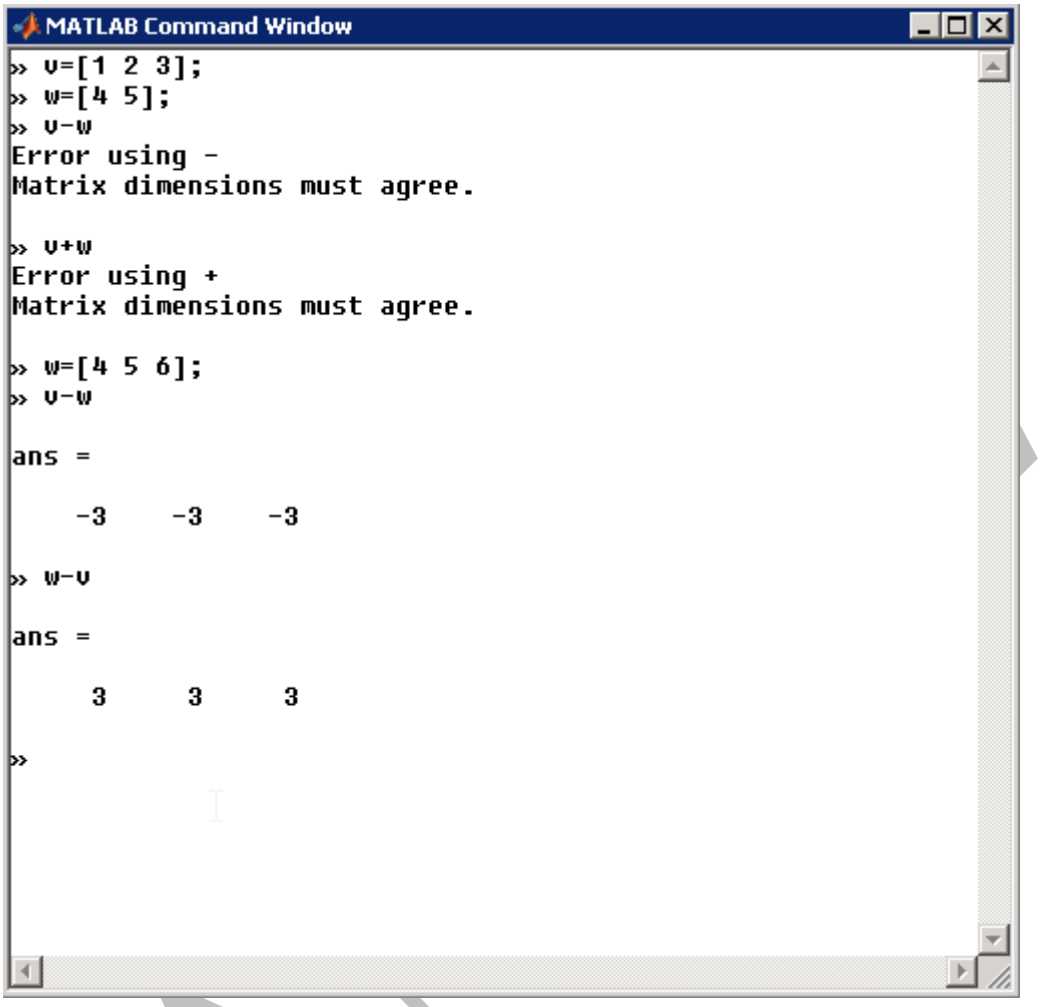
Something also very very important is to be able to create vector ranges using **linspace( )** function or not:



Remember that you can't put string in vectors but only in a cell array that's why the definition of the first use **[ ]** and the second **{ }**:

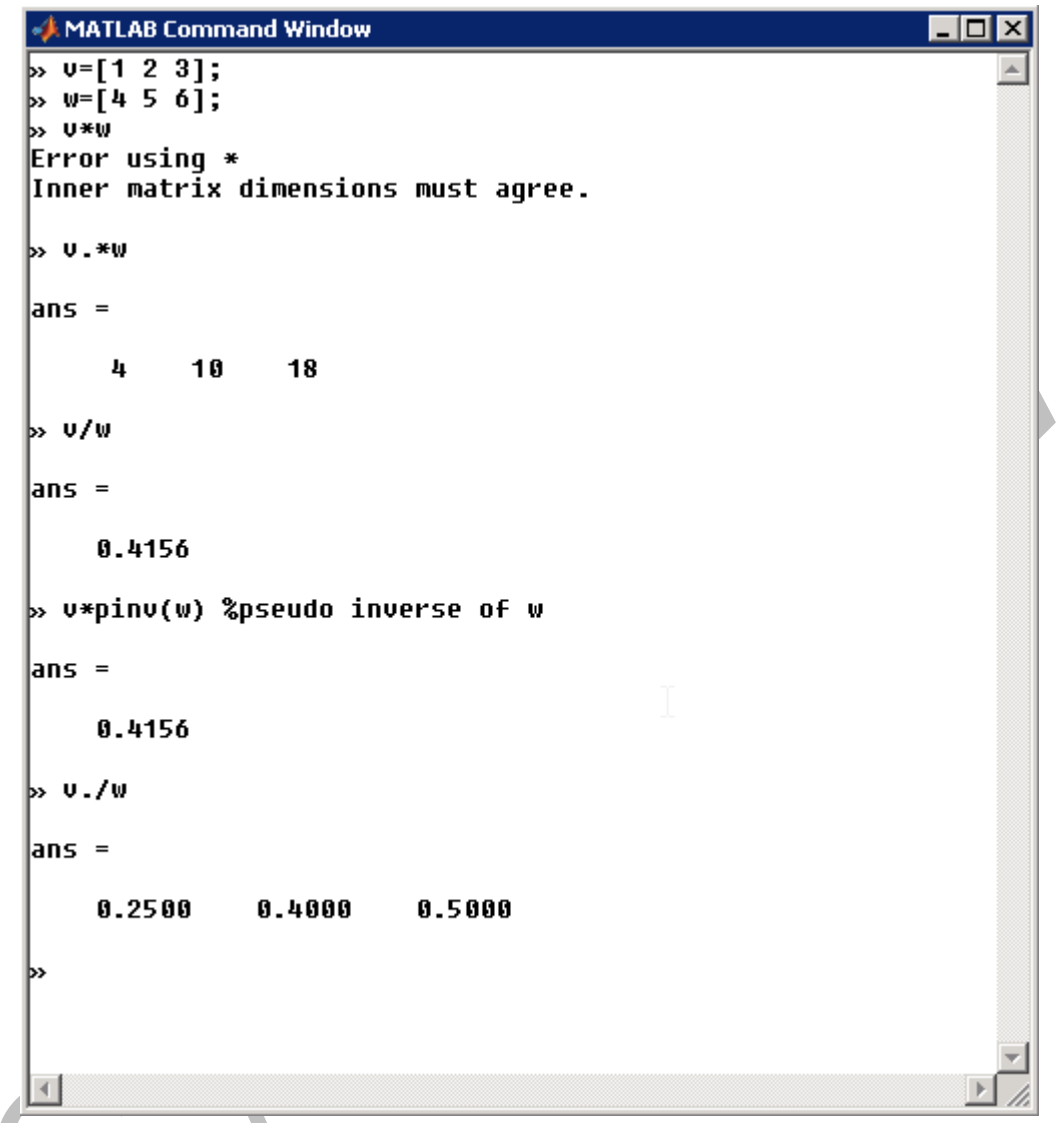


You always have to respect vectors dimensions rules that you have seen during your study in vector analysis and linear algebra even in MATLAB ;-P :

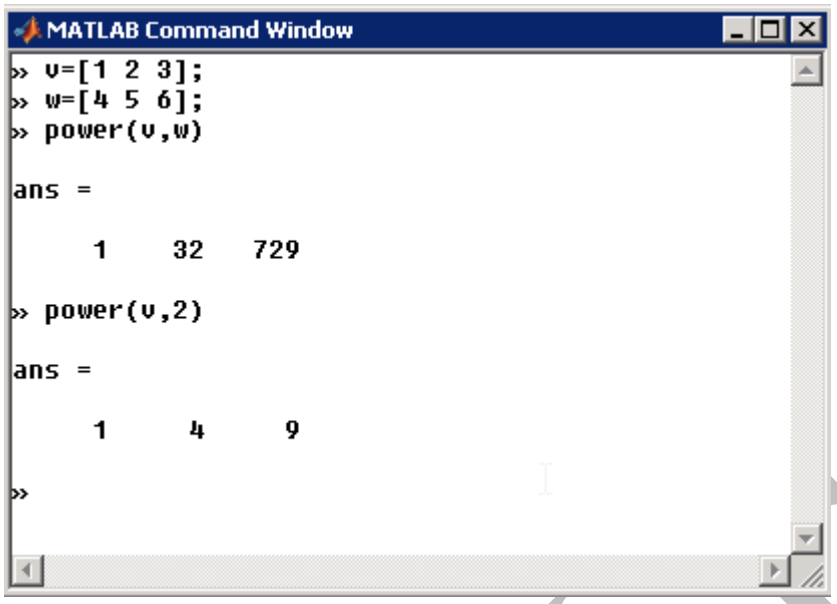


SAMPLE

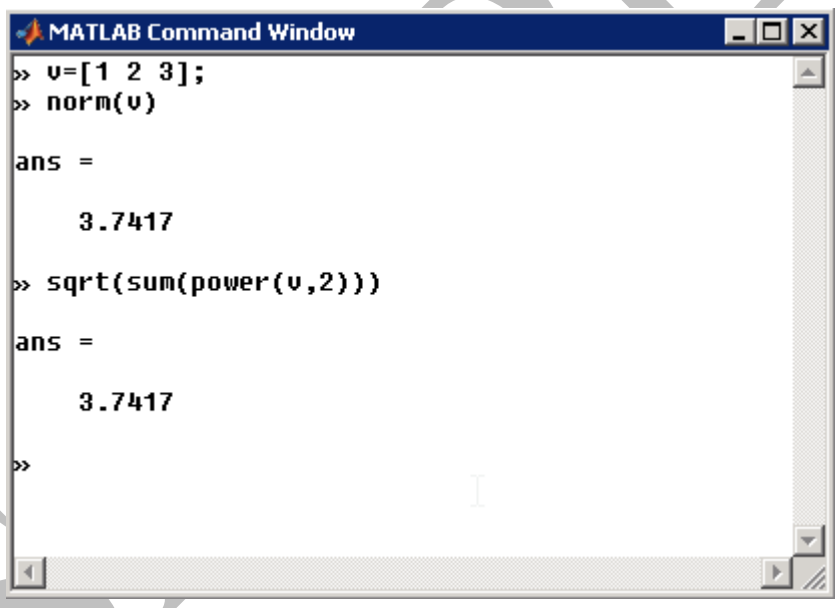
There is something very important to note: in MATLAB if you precede the multiplication symbol, division symbol or power symbole by a point (**.***, **./**, **.^X**) it means that you are doing a component by component operation:



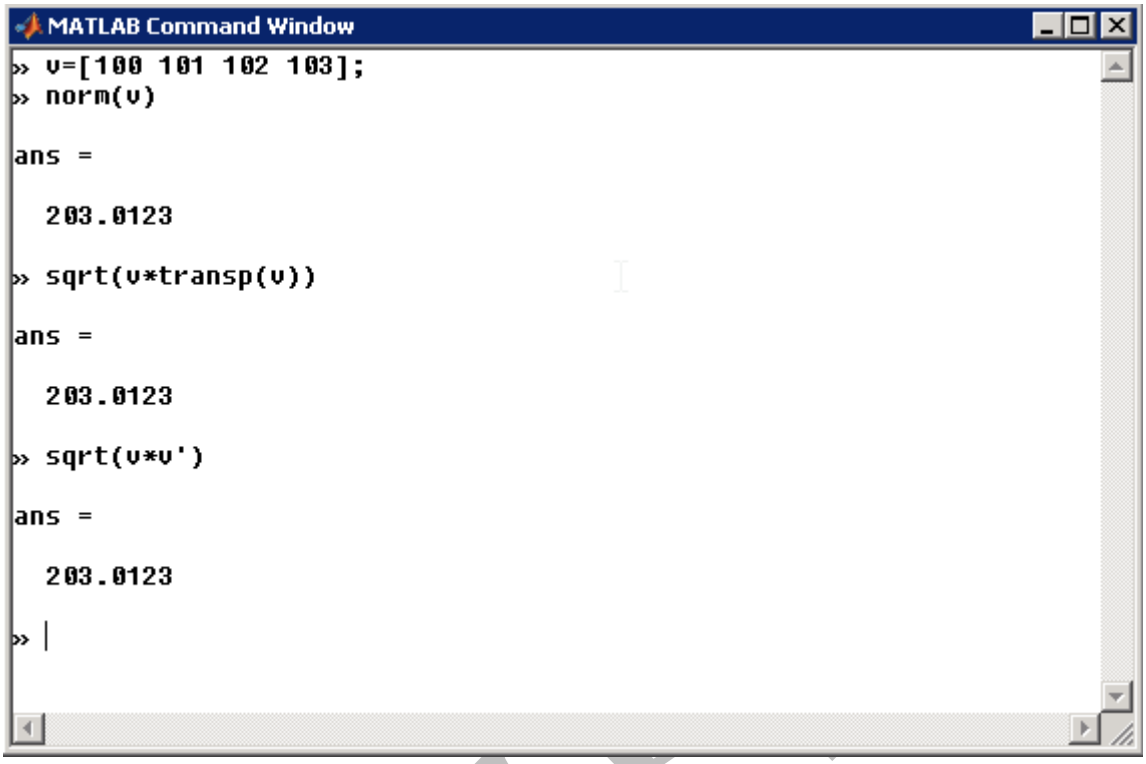
The **power( )** function apply to a vector with a vector as exponent is equivalent to a component by component **.^X** power:



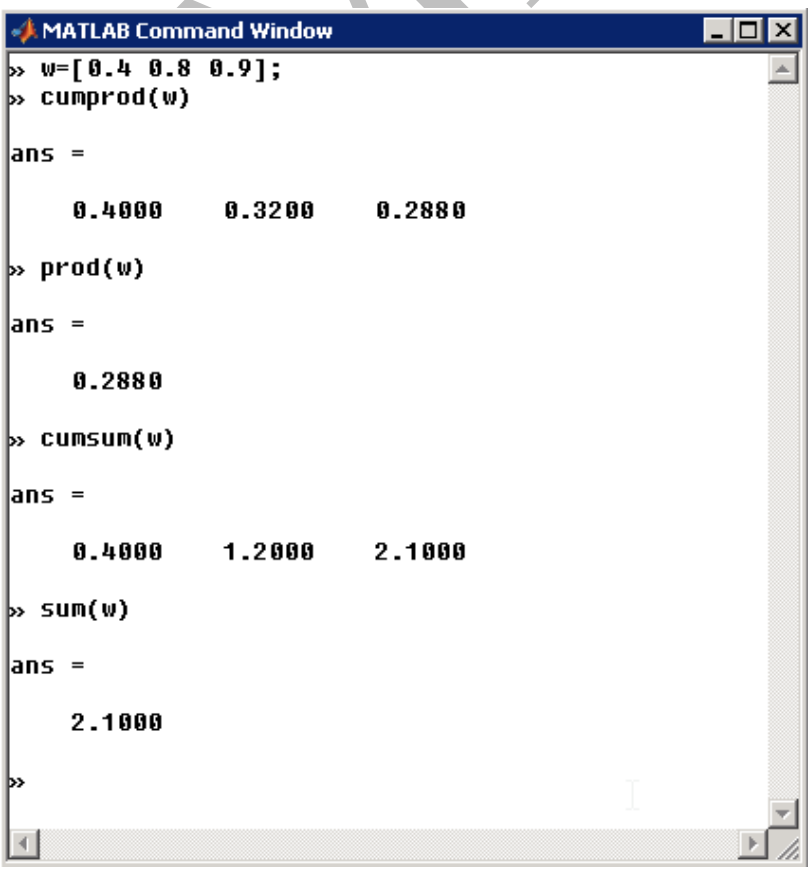
We also have the function **norm( )** to calculate the Euclidian norm of a vector (we make also quickly the check...):



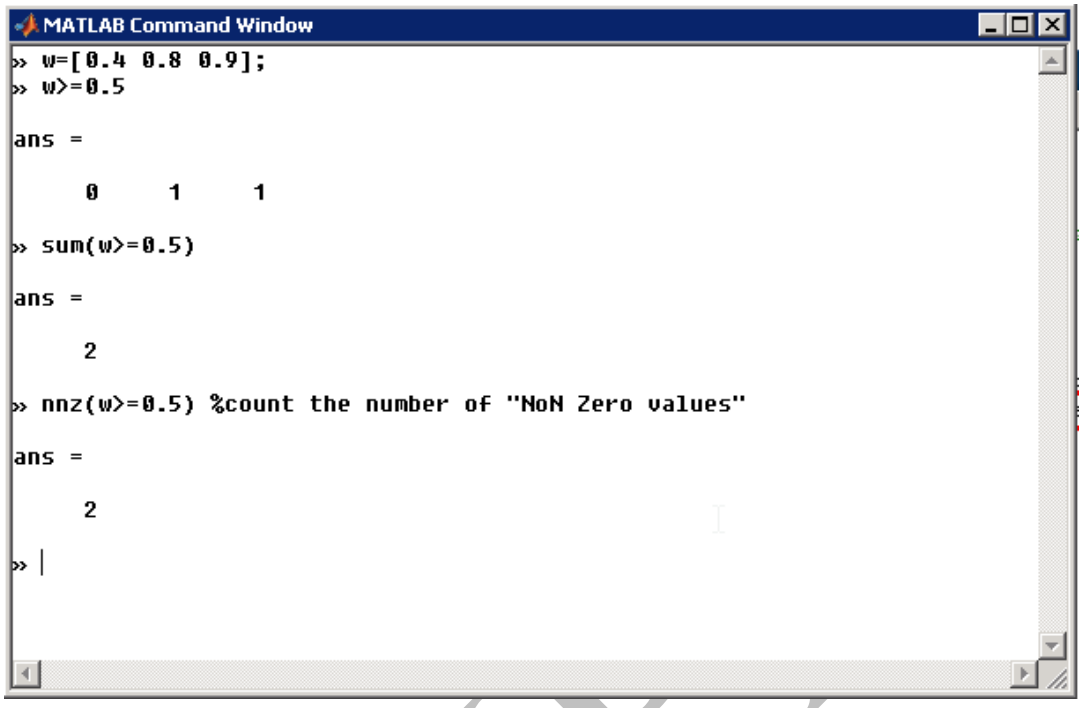
Or some other possible syntax for the norm that you can sometimes see in scripts:



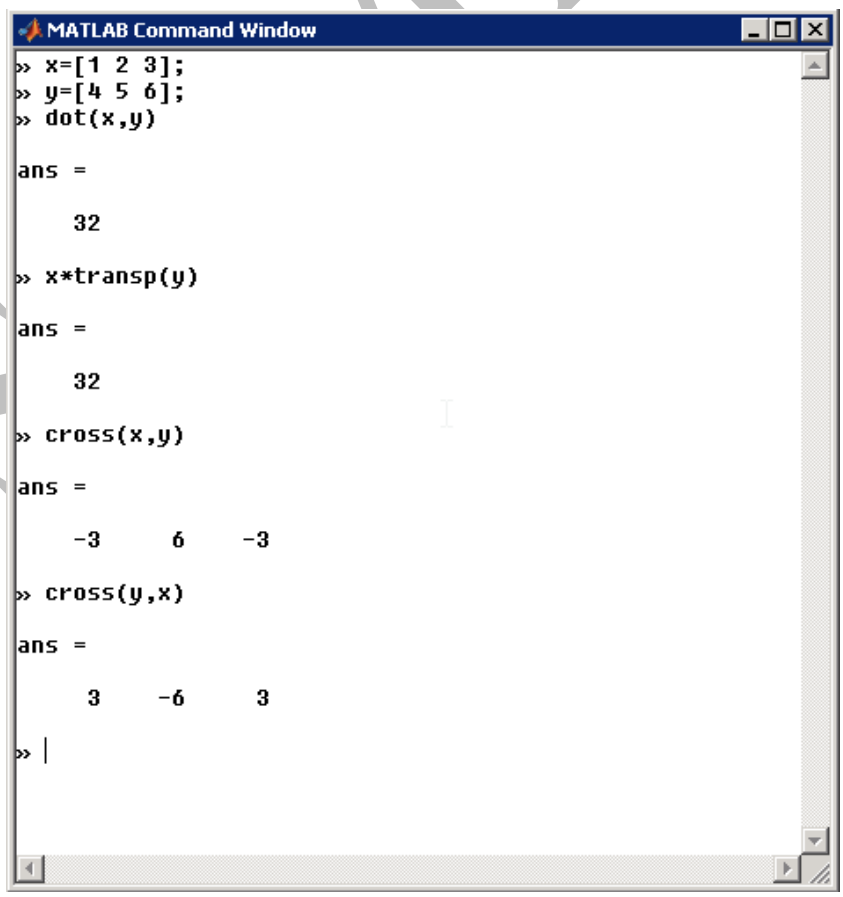
If you work with probabilities vector the **cumprod( )** function will be very useful for you, and if you work in the field of finance you will appreciate the **cumsum( )** function:



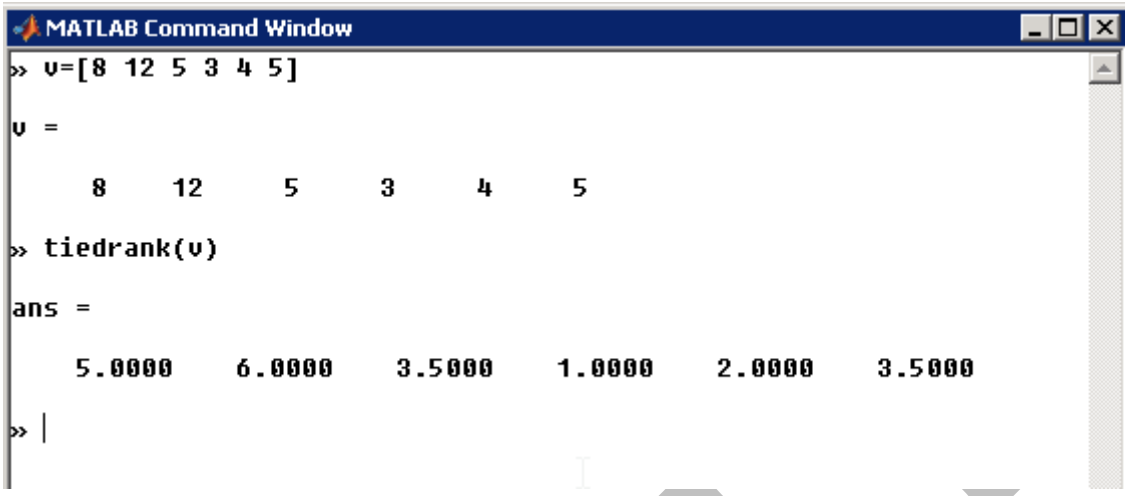
We can also apply logical operations on vectors and especially for example non zero values with the function **nnz( )**:



We can also have a look at the dot product and cross product with the functions **dot( )** and **cross( )** that are very useful in physics:

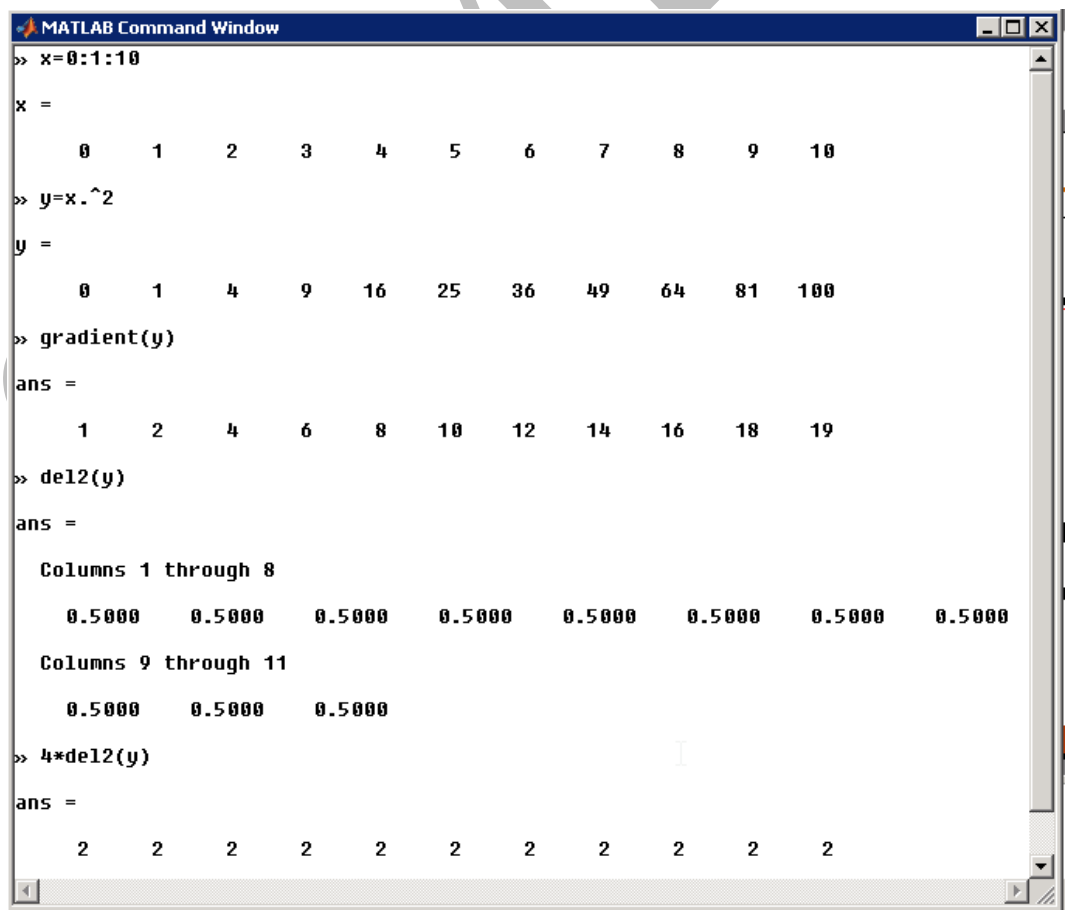


You certainly remember that the function **unique( )** returns also the rank, but there is also a function named **tiedrank( )** that returns another type of ranking method keeping all original values:



Now take a look the **gradient( )** function by checking that it's effectively giving back the derivate and the laplacian with the function **del2( )** giving back the second order derivate with a simple naive example.

As you can see below **there is a trap** with MATLAB when you use **del2( )**. You have to multiply the result by 4 to get the standard laplacian:



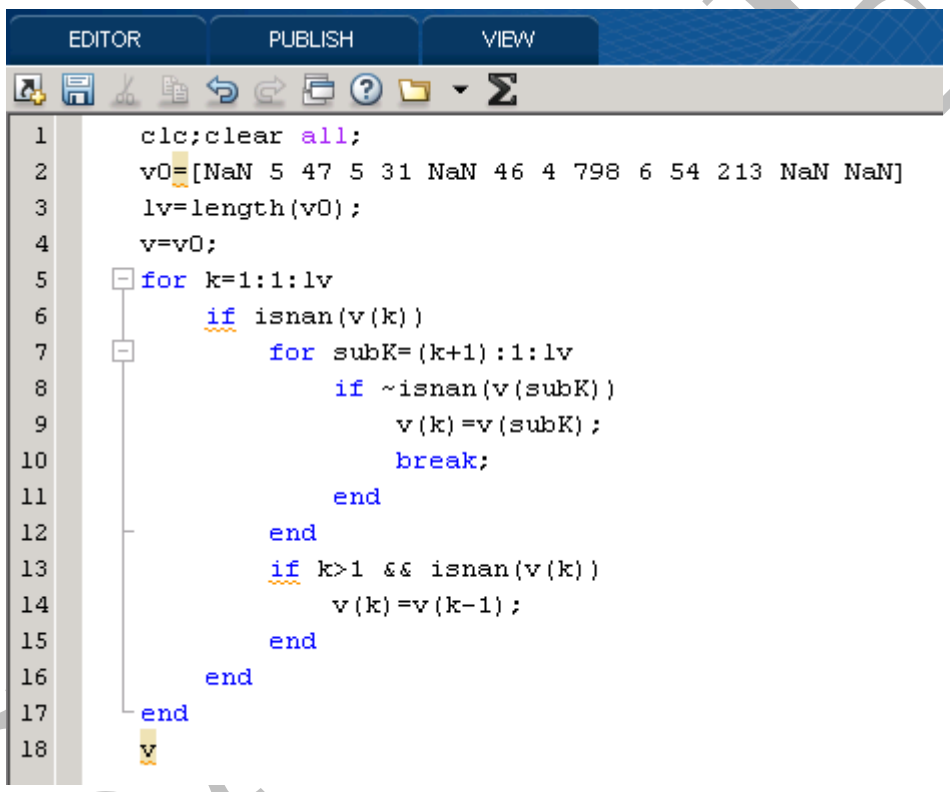
Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seems to have built-in functions.

Clean a vector

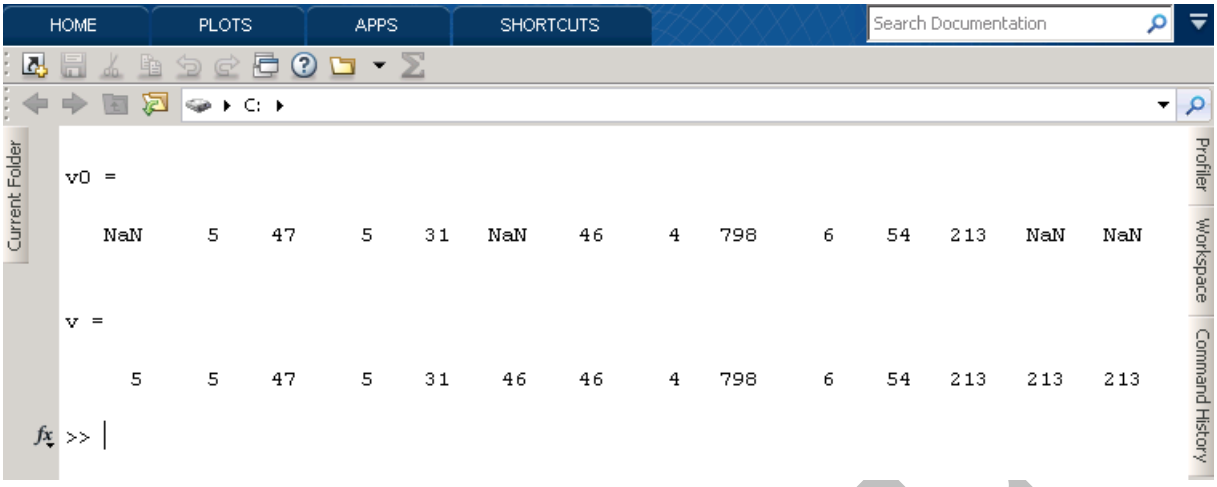
Consider we have a return vector retrieved from a financial feed. Some values can be missing. In case of missing value at the beginning, the rule is to take the next available one... otherwise we take the previous day value.

Consider this practical case as a challenge! The fastest code written today by one of my student is the following:




```
clc;clear all;
v0=[NaN 5 47 5 31 NaN 46 4 798 6 54 213 NaN NaN]
lv=length(v0);
v=v0;
for k=1:1:lv
    if isnan(v(k))
        for subK=(k+1):1:lv
            if ~isnan(v(subK))
                v(k)=v(subK);
                break;
            end
        end
        if k>1 && isnan(v(k))
            v(k)=v(k-1);
        end
    end
end
v
```

This gives:



Build an orthonormal vector base

So, given vector

$$v = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

in $\mathbb{R}^3$, it follows that any point (x, y, z) that satisfies:

$$ax + by + cz = 0$$

(i.e. a point that lives on plane whose normal vector is *v*) will define components of a vector that *u* is orthogonal to *v*.

One such vector is:

$$u = \begin{pmatrix} 0 \\ -c \\ b \end{pmatrix}, u = \begin{pmatrix} -c \\ 0 \\ a \end{pmatrix}$$

you can choose whatever you prefer, but it cannot be:

$$u = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

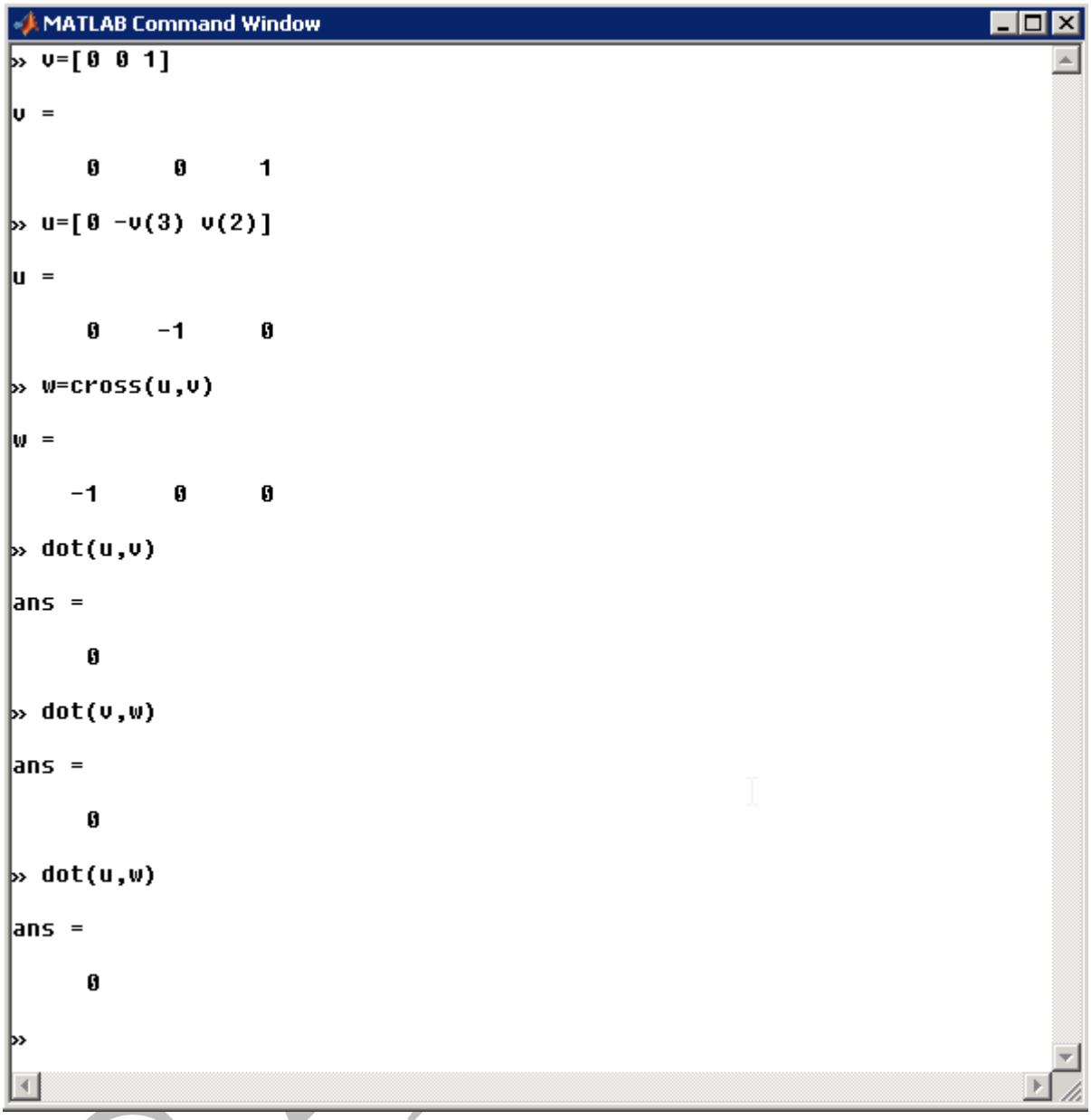
Now, given *u* and *v*, compute the third vector that is orthogonal to both *u* and *v* with

$$w = u \times v$$

To make an orthonormal basis just divide each vector by its norm!

Demo:

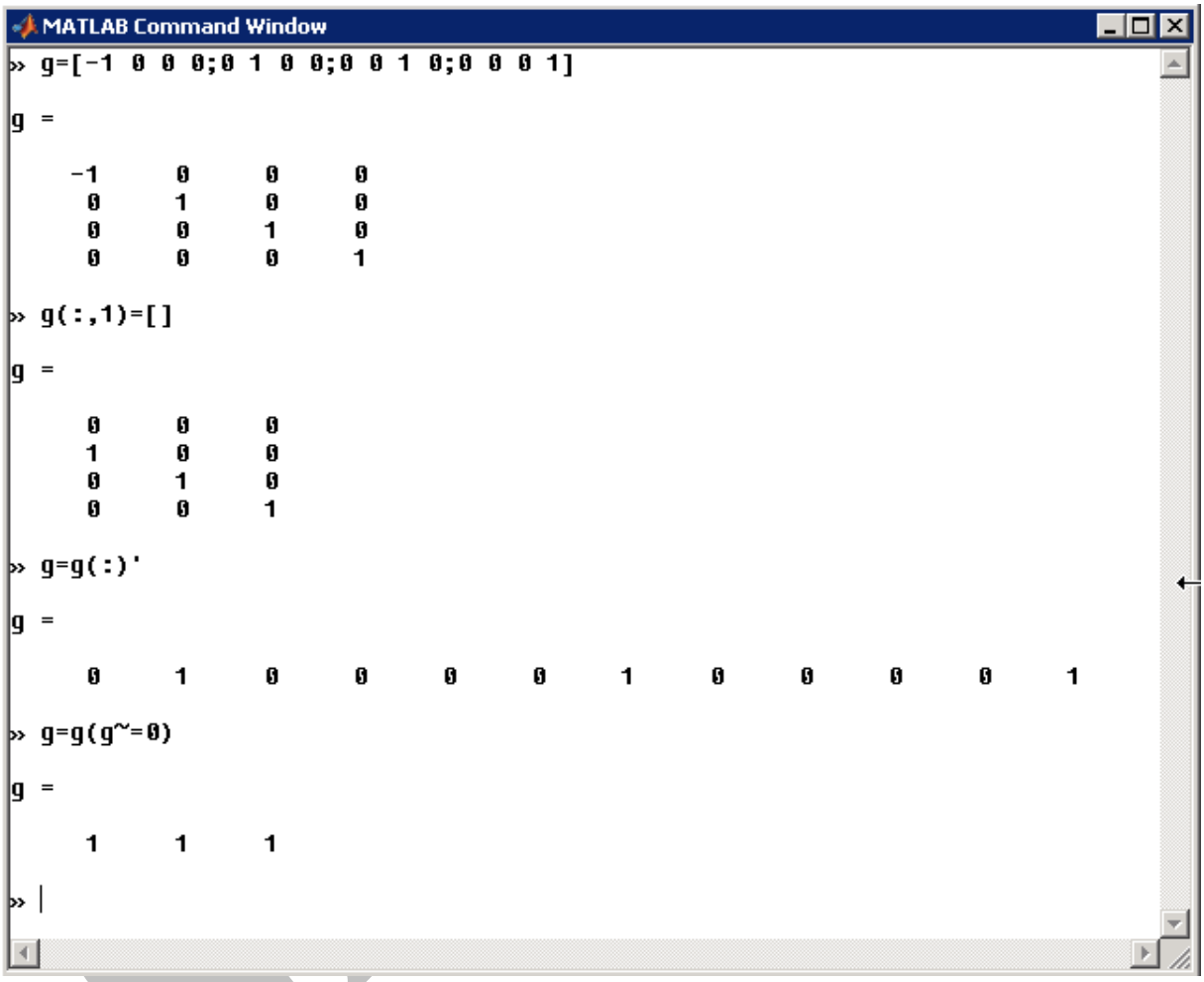
SAMPLE


MATLAB Command Window
» v=[0 0 1]
v =
0 0 1
» u=[0 -v(3) v(2)]
u =
0 -1 0
» w=cross(u,v)
w =
-1 0 0
» dot(u,v)
ans =
0
» dot(v,w)
ans =
0
» dot(u,w)
ans =
0
»

Arithmetic operations and manipulations of matrices

Master the manipulation of matrices is even more important than vectors. Let us see some manipulations with this type of objects (typical matrix algebra)

We begin something simple with the Minkowsky matrix (see chapter of Relativity of my applied mathematics e-book) by removing some parts of it and getting the diagonal by excluding first the usage of the **diag( )** function (see next page) :



Now let us see that we can use the **diag( )** function or **trace( )** function that are very useful in advanced physics and also in statistics (when you build your own confusion matrix):

```
MATLAB Command Window
» g=[-1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]

g =

    -1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

» diag(g)

ans =

    -1
     1
     1
     1

» sum(diag(g))

ans =

     2

» trace(g)

ans =

     2

»
```

Sometimes it can be useful to generate a diagonal unit matrix. You can do that very easily with the **eye( )** function:

```
MATLAB Command Window
» M=eye

M =

     1

» M=eye(2)

M =

     1     0
     0     1

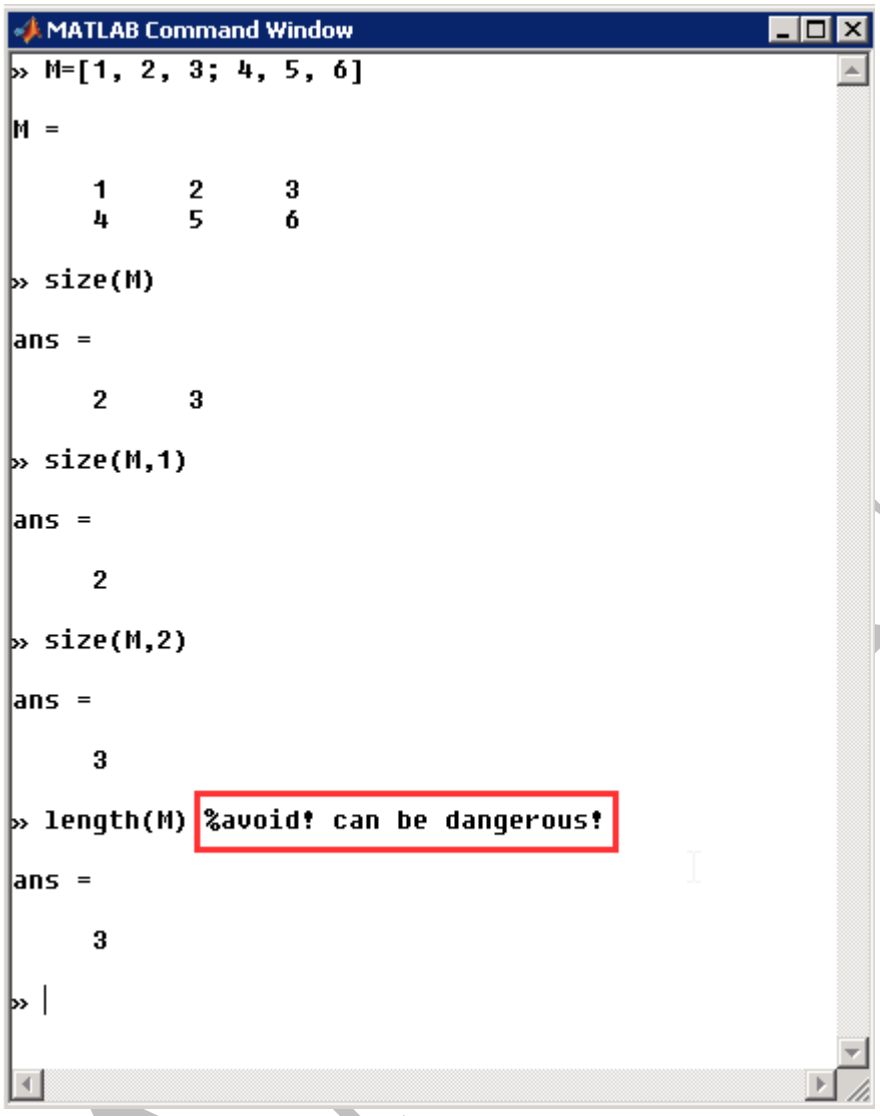
» M=eye(3)

M =

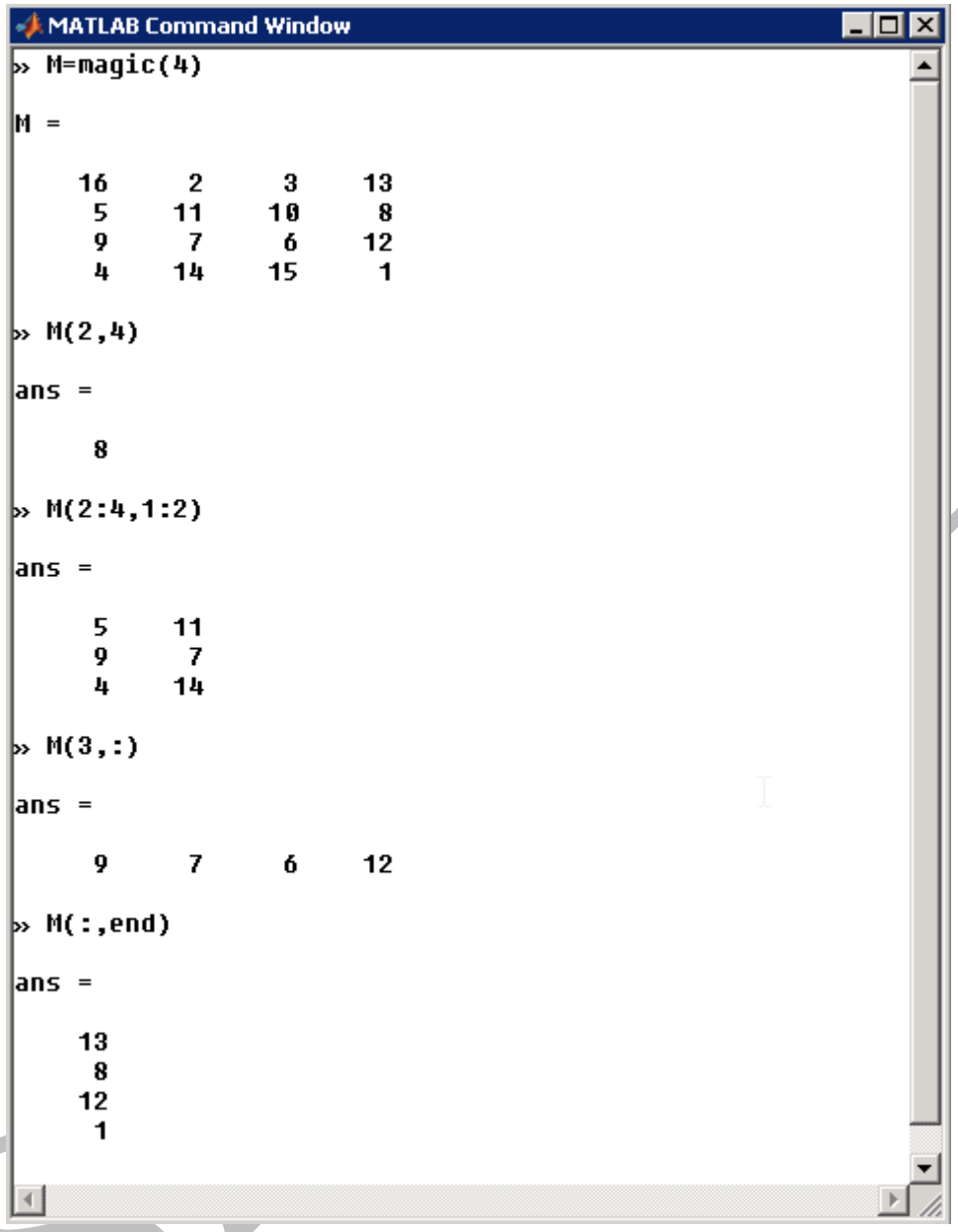
     1     0     0
     0     1     0
     0     0     1

»
```

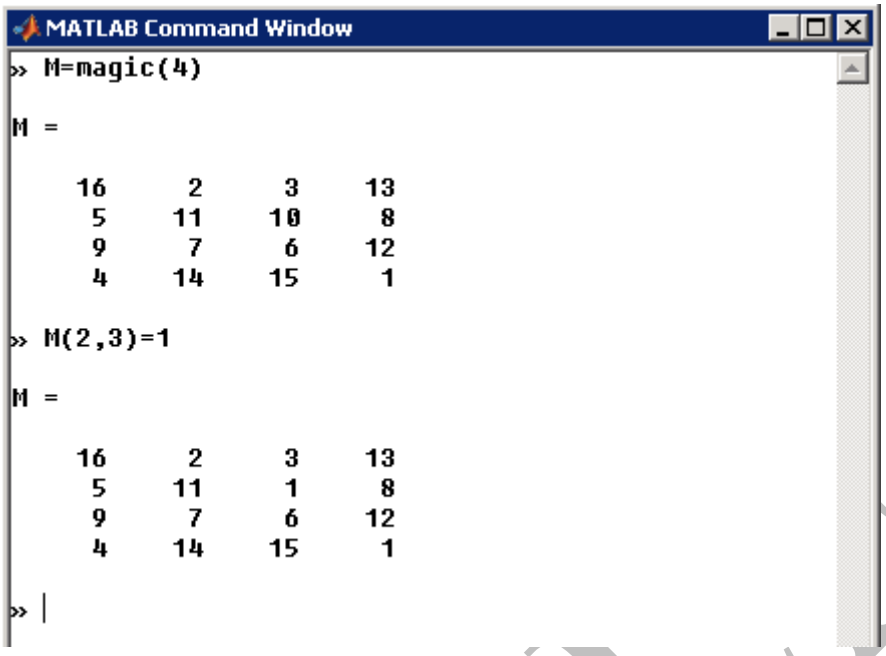
Now have a look why it is **dangerous** to use the **length( )** function instead of the **size( )** function to get dimensions of a matrix:


MATLAB Command Window
» M=[1, 2, 3; 4, 5, 6]
M =
1 2 3
4 5 6
» size(M)
ans =
2 3
» size(M,1)
ans =
2
» size(M,2)
ans =
3
» length(M) %avoid! can be dangerous!
ans =
3
»

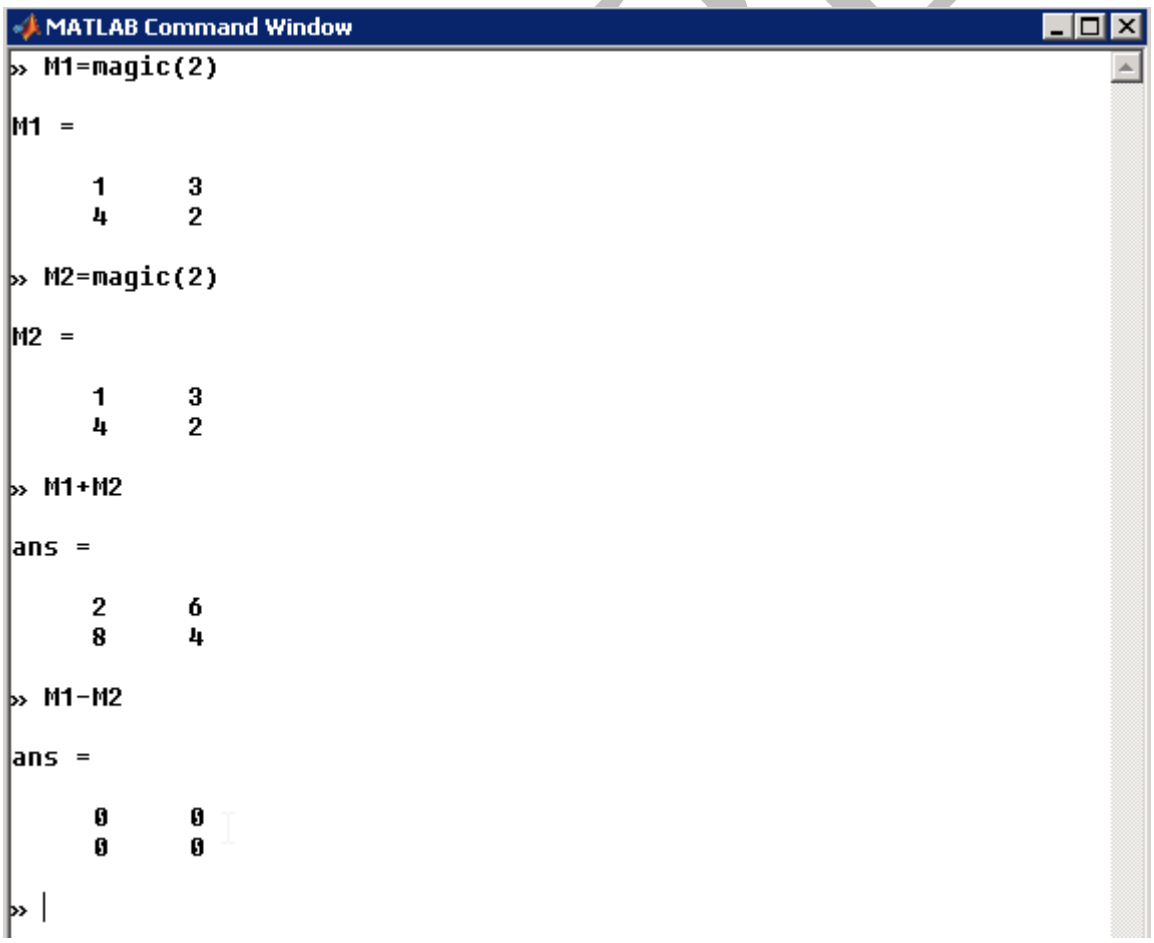
And now some stuff about extracting some parts of a matrix:



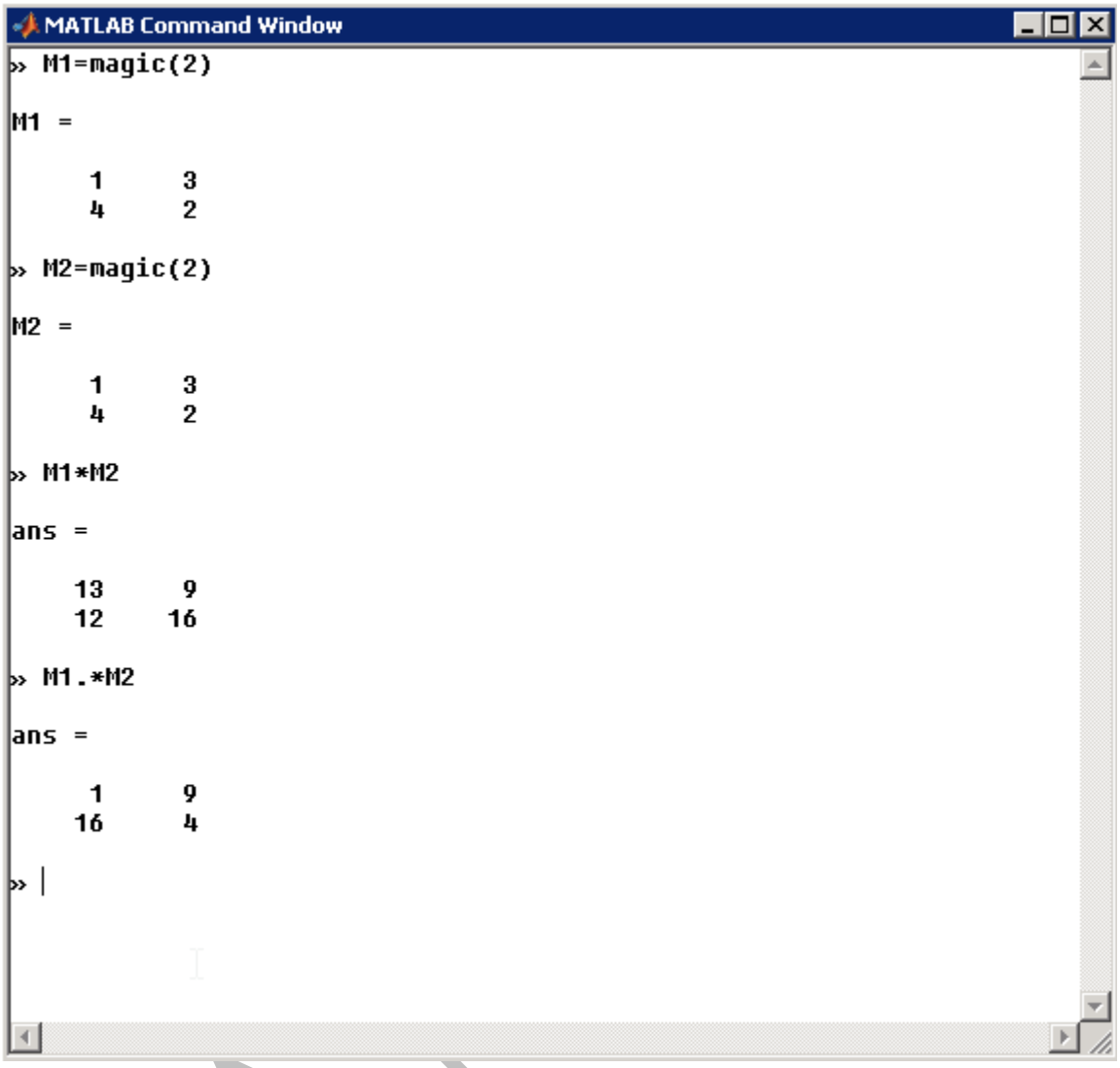
Or change the value of a given component:



Otherwise we can check that adding matrices of the same dimension works well:

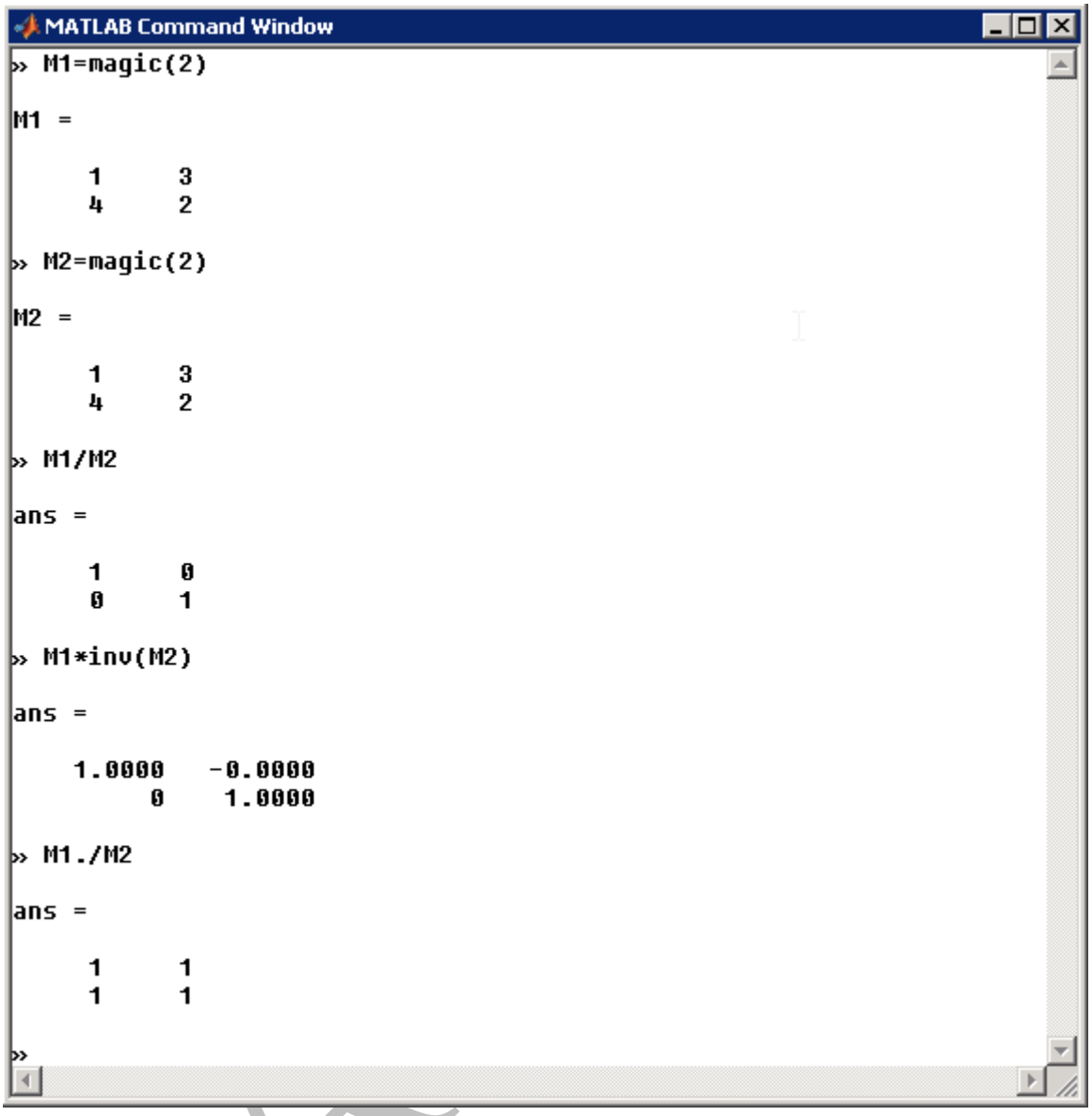


And also matrices multiplication if you respect linear algebra rules or that you can also make a component by component multiplication like for vectors:



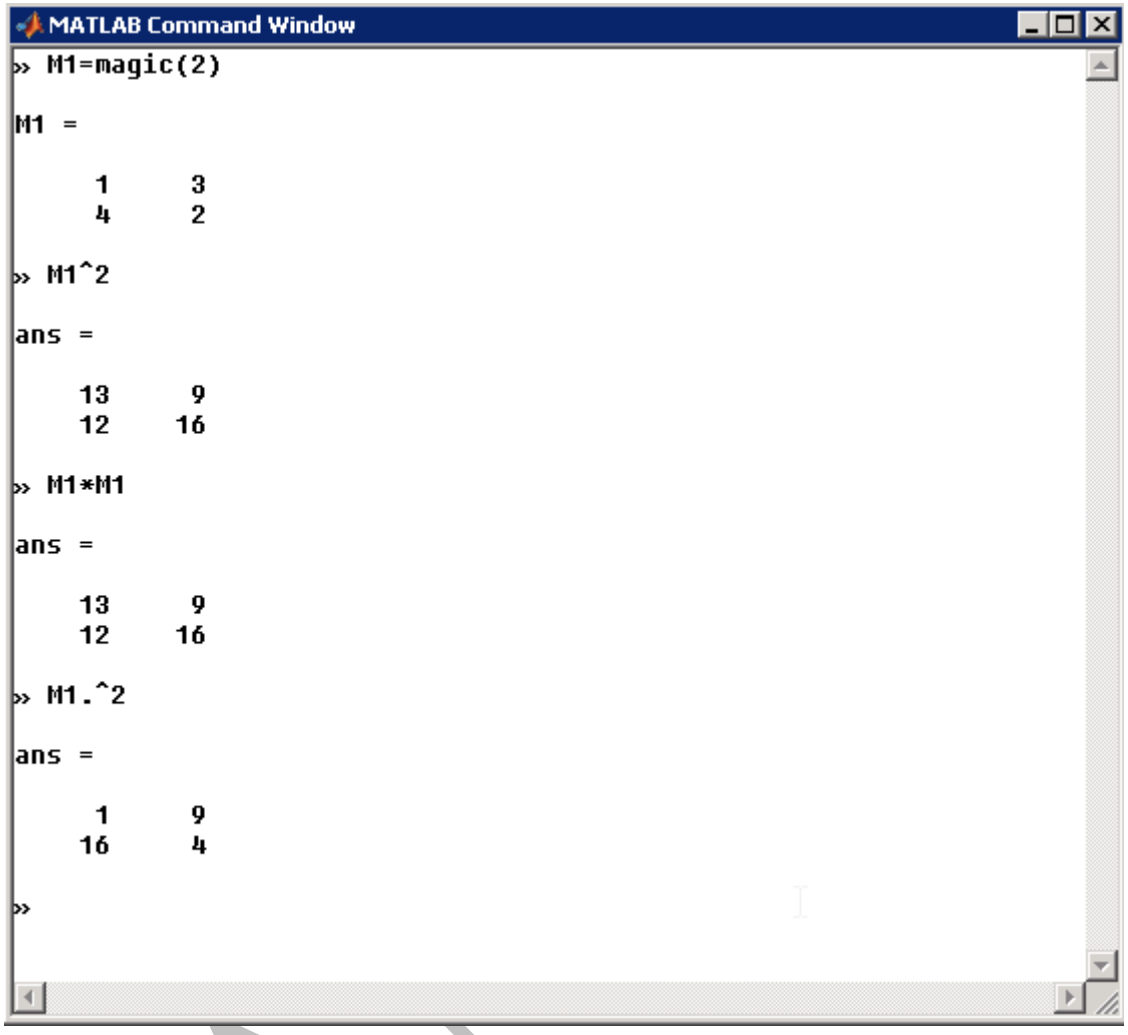
We can also take a look to what happens with division and if it's corresponding with the inverse trough the function **inv( )**:

SAL



Everything seems all right and standard.

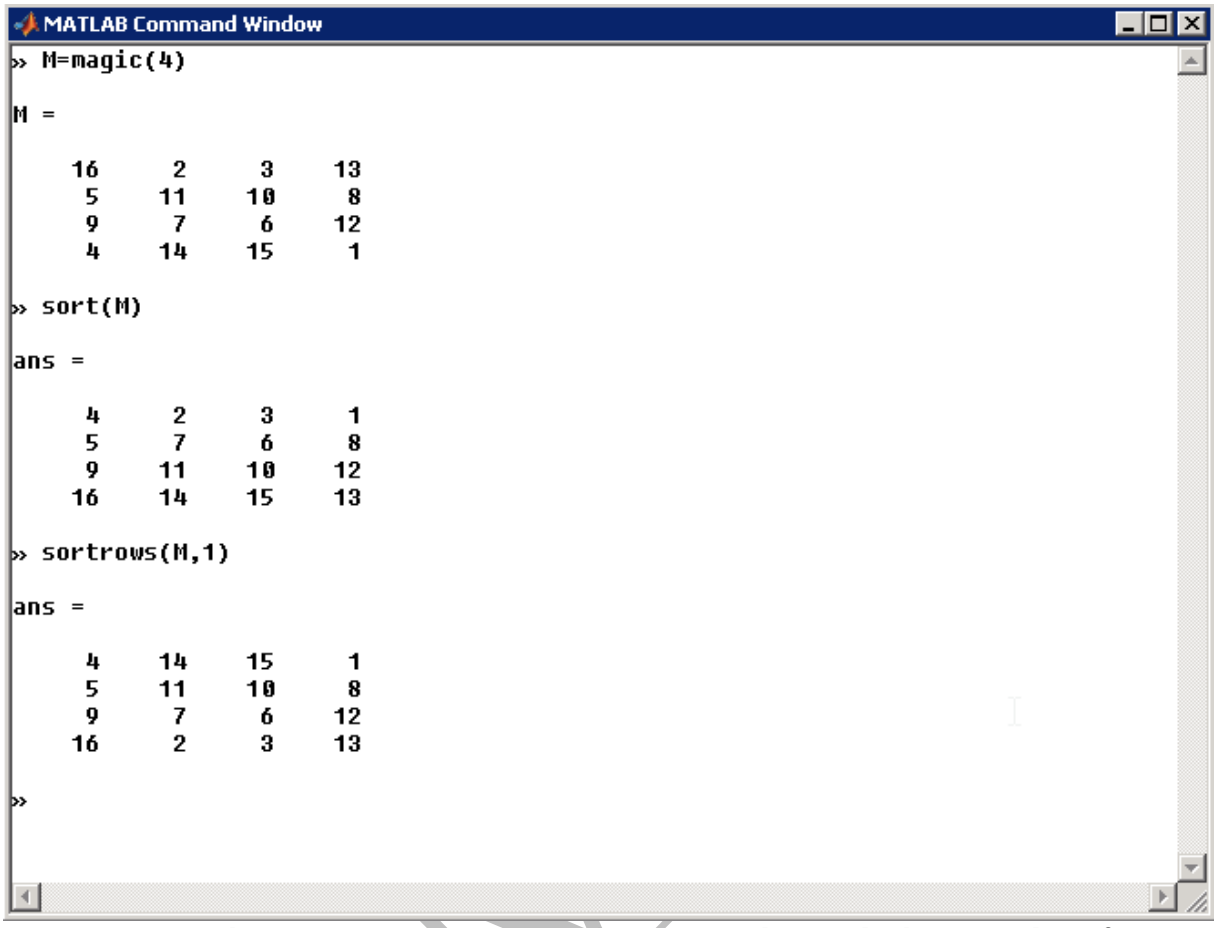
Let's have a look with multiplication and power to see if everything is also standard:



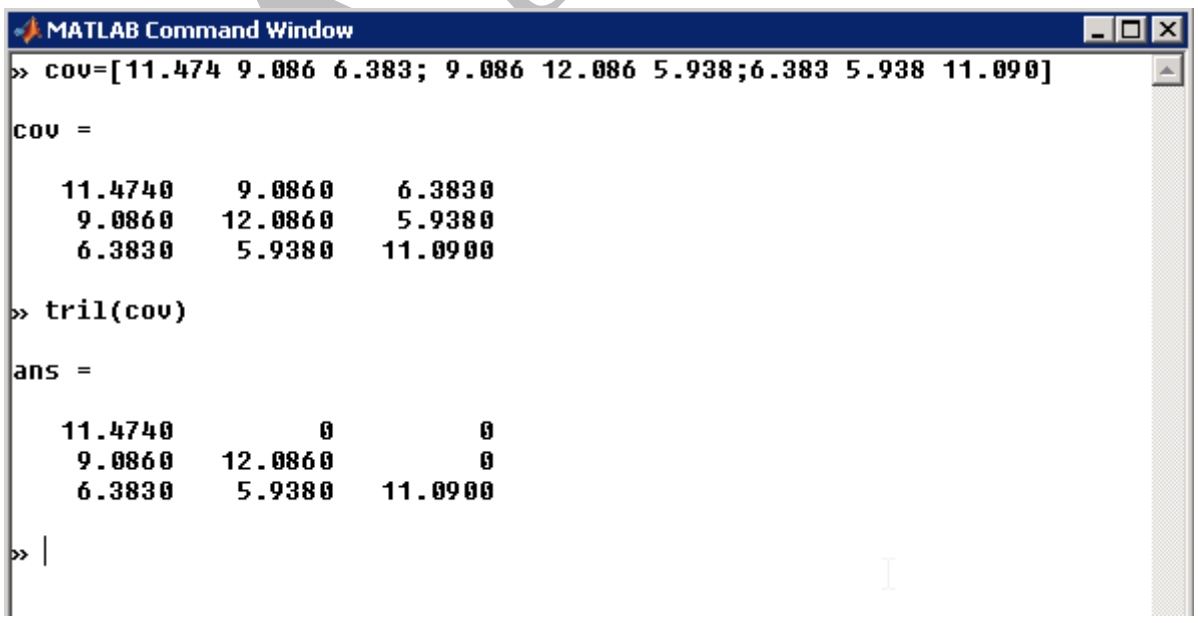
Yep! Everything seems fine (standard)!

SAM

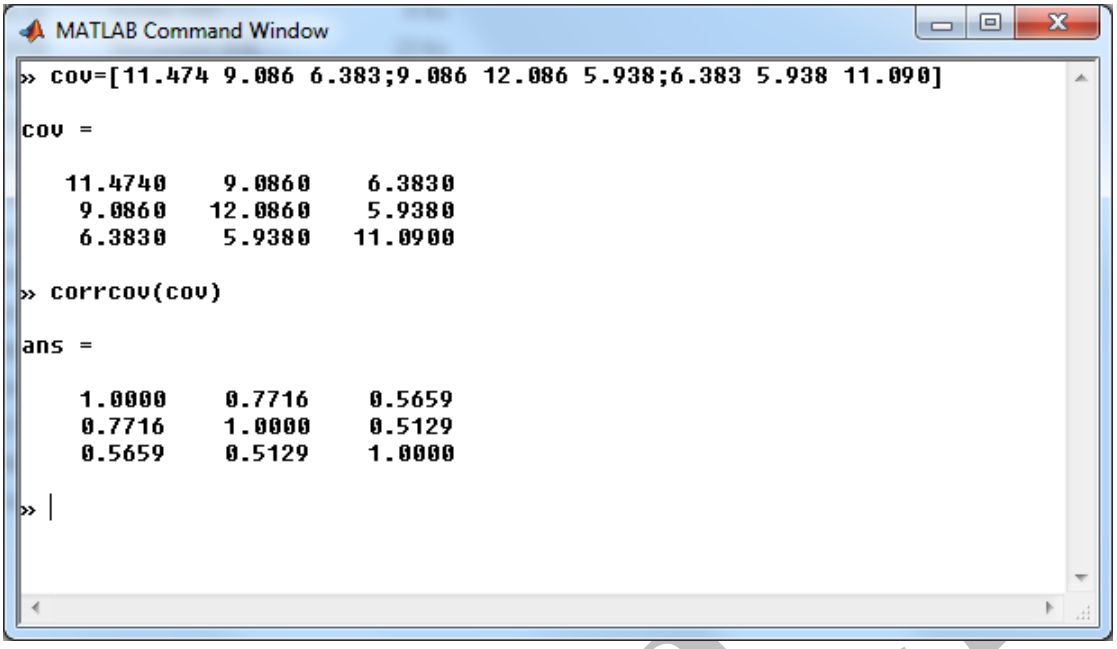
And what about sorting a matrix by columns with the function **sort( )** or by columns with the function **sortrows( )**:



Now suppose we have a covariance matrix and we want to keep only the triangular inferior part. Thus we can use the function **tril( )**:



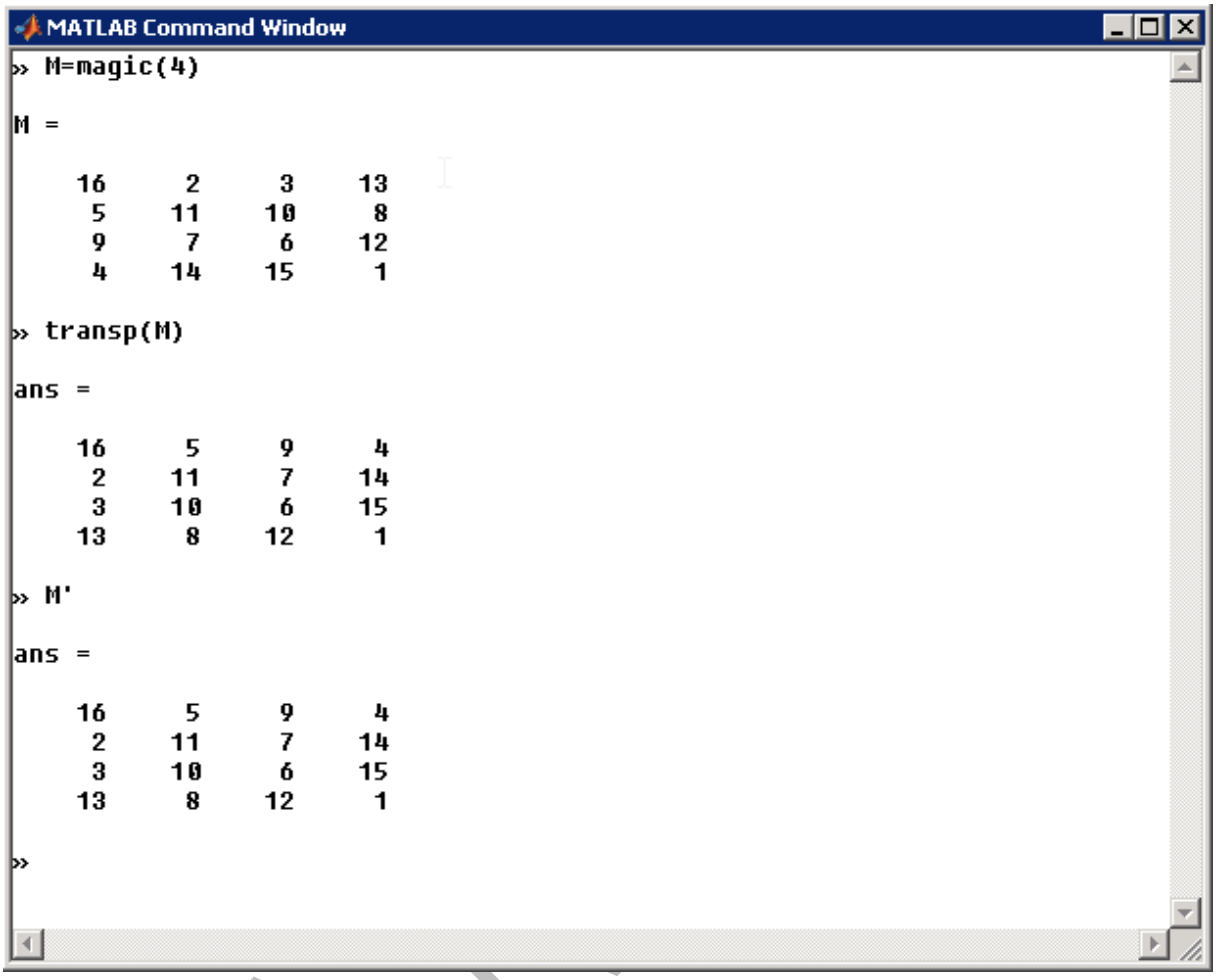
We know we need sometimes in quantitative finance to convert a covariance matrix in a correlation matrix. This can easily be done with the function **corrcov( )**:



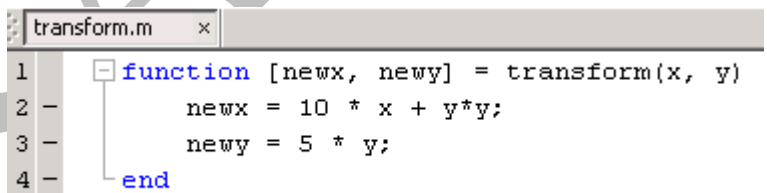
And as we can check it very easily by hand, the conversion is correct!

SAMPLE

And have also a check to the tradition transposition with the function **transp( )** or the apostroph ':

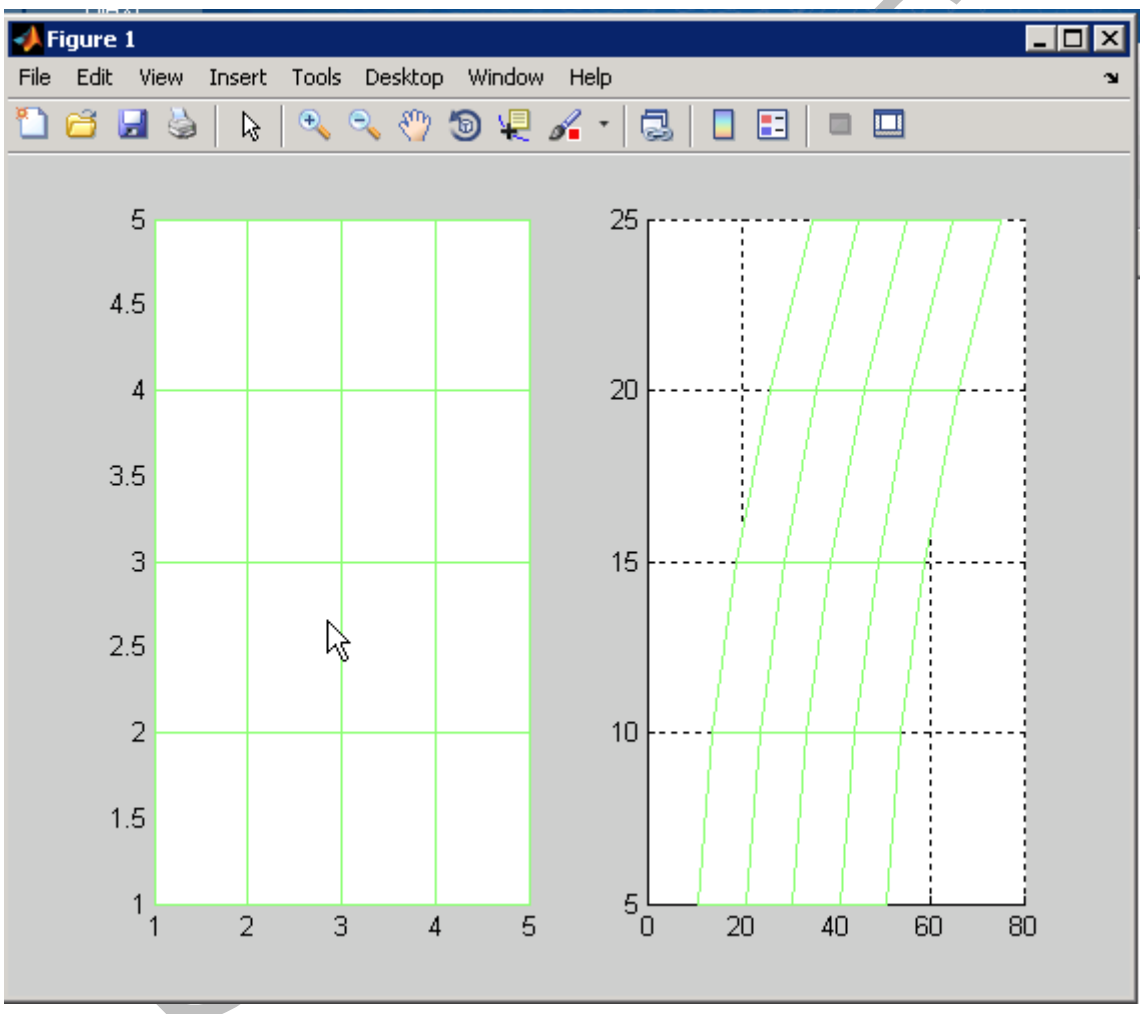


As we know at the high school level matrix can be used as operator on points. Let us see how to create such a transformation. First create the following function:



After if we write we get:

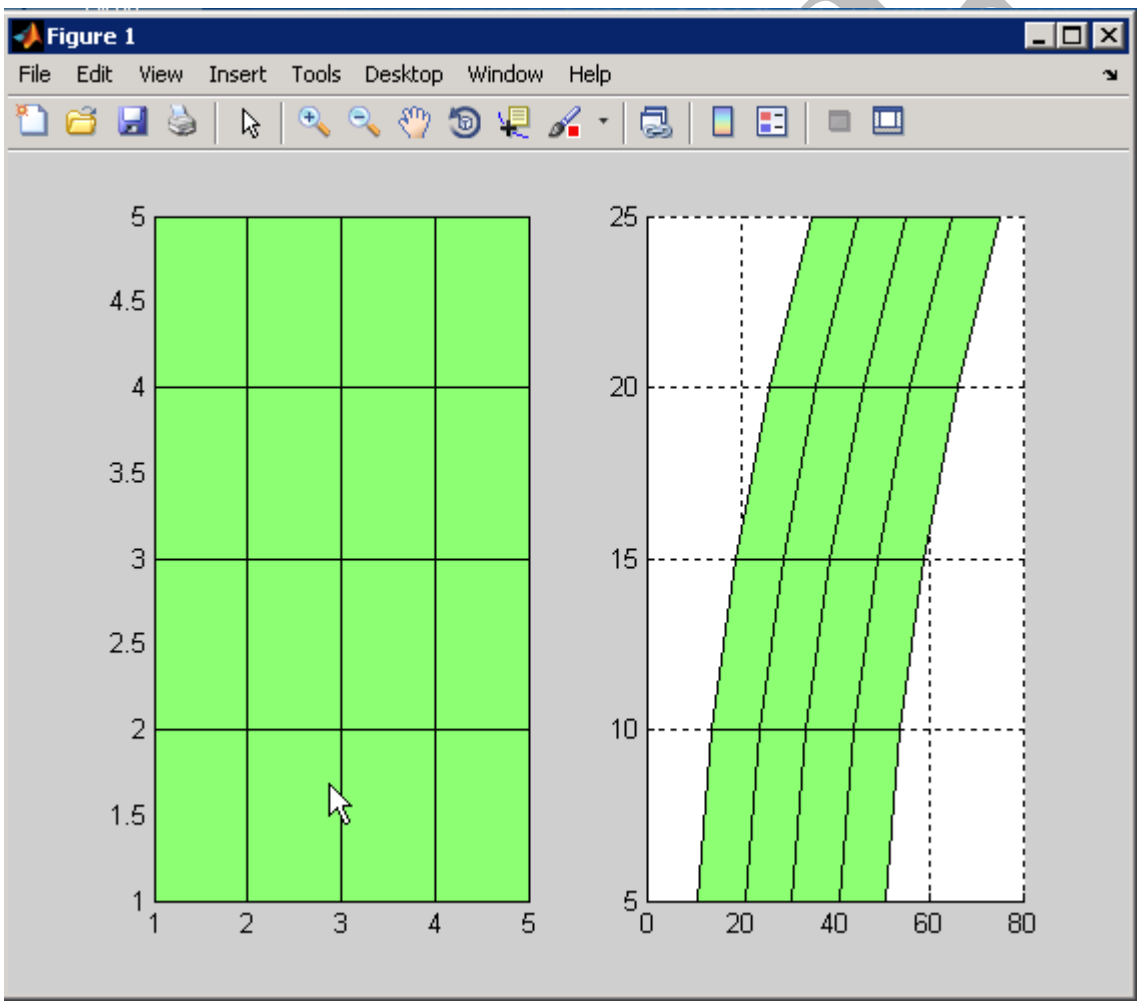
```
1     [x, y] = meshgrid(1:5, 1:5);
2     subplot(1,2,1)
3     mesh(x, y, zeros(size(x)));
4     view(0,90);
5     subplot(1,2,2)
6     %transformed data
7     [u, v] = arrayfun(@transform, x, y);
8     mesh(u, v, zeros(size(u)))
9     view(0,90)
```



Or:

```
1     [x, y] = meshgrid(1:5, 1:5);
2     subplot(1,2,1)
3     surf(x, y, zeros(size(x)));
4     view(0,90);
5     subplot(1,2,2)
6     %transformed data
7     [u, v] = arrayfun(@transform, x, y);
8     surf(u, v, zeros(size(u)))
9     view(0,90)
```

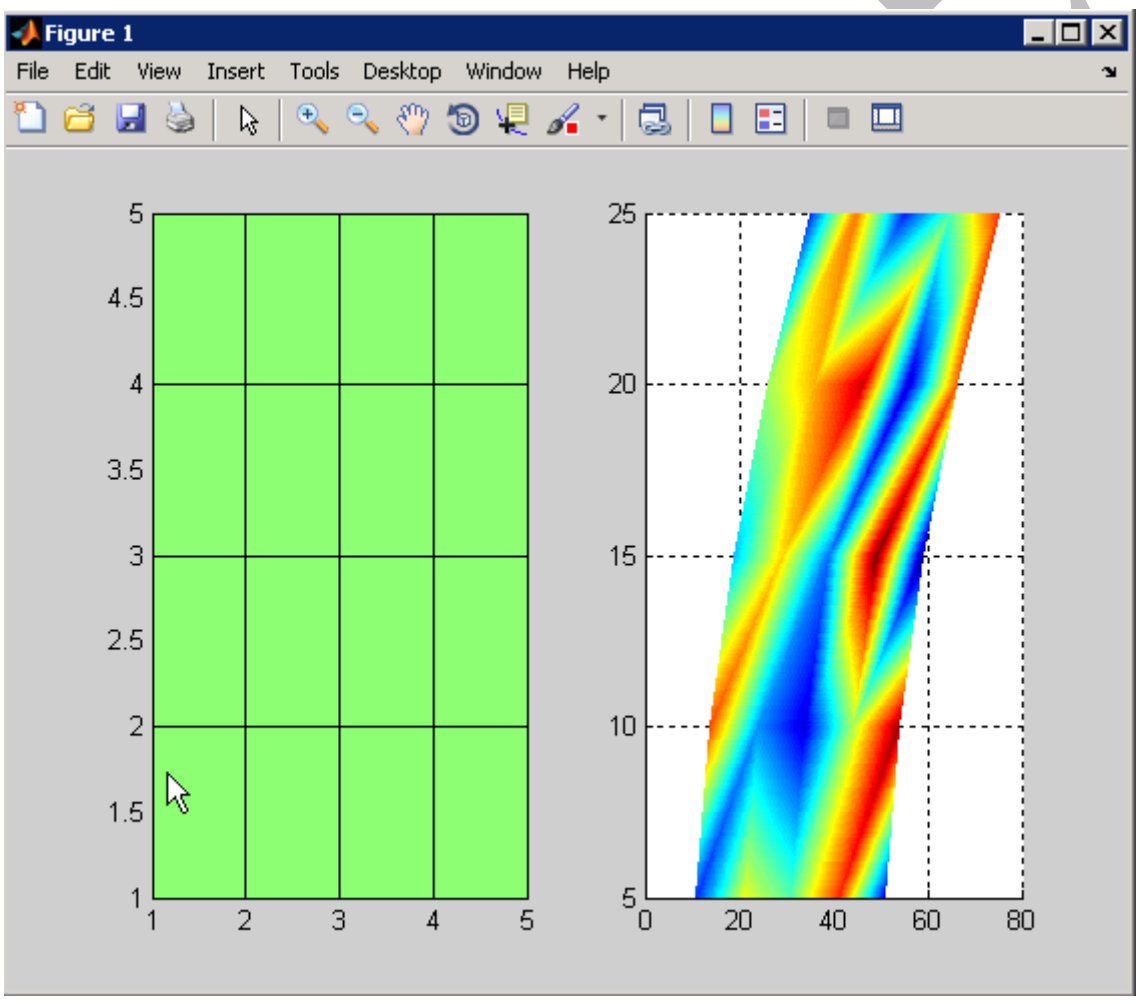
So we get:



Or even better (this will give some idea to people that made engineering studies in materials constraints):

```
1     [x, y] = meshgrid(1:5, 1:5);
2     subplot(1,2,1)
3     surf(x, y, zeros(size(x)));
4     view(0,90);
5     subplot(1,2,2)
6     %transformed data
7     [u, v] = arrayfun(@transform, x, y);
8     surf(u, v, rand(size(u)))
9     shading interp
10    view(0,90)
```

so we get:

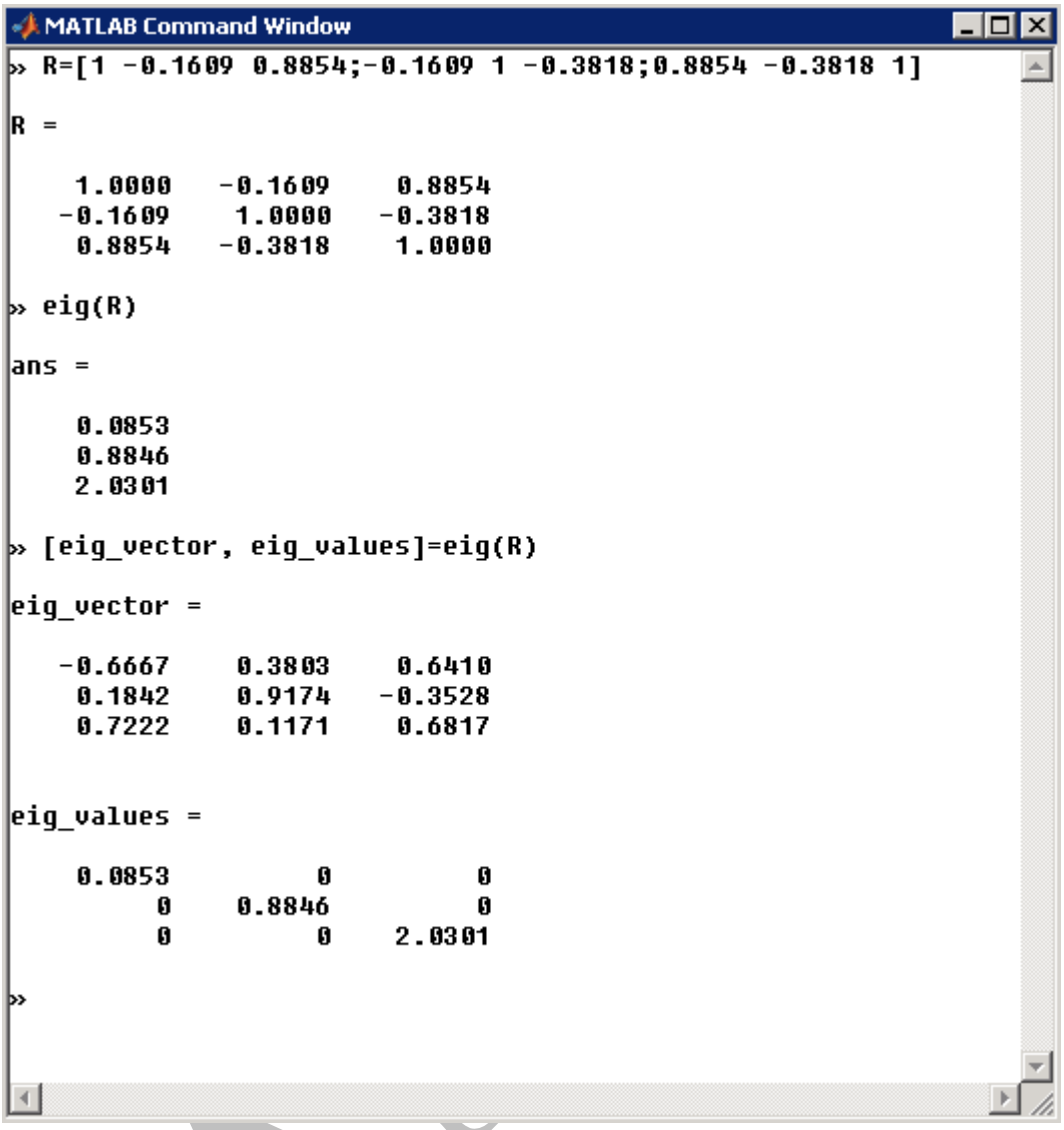


And we can also check that the inverse and determinant using the function **det( )** are corresponding to what we have calculate by hand in the theoretical course:



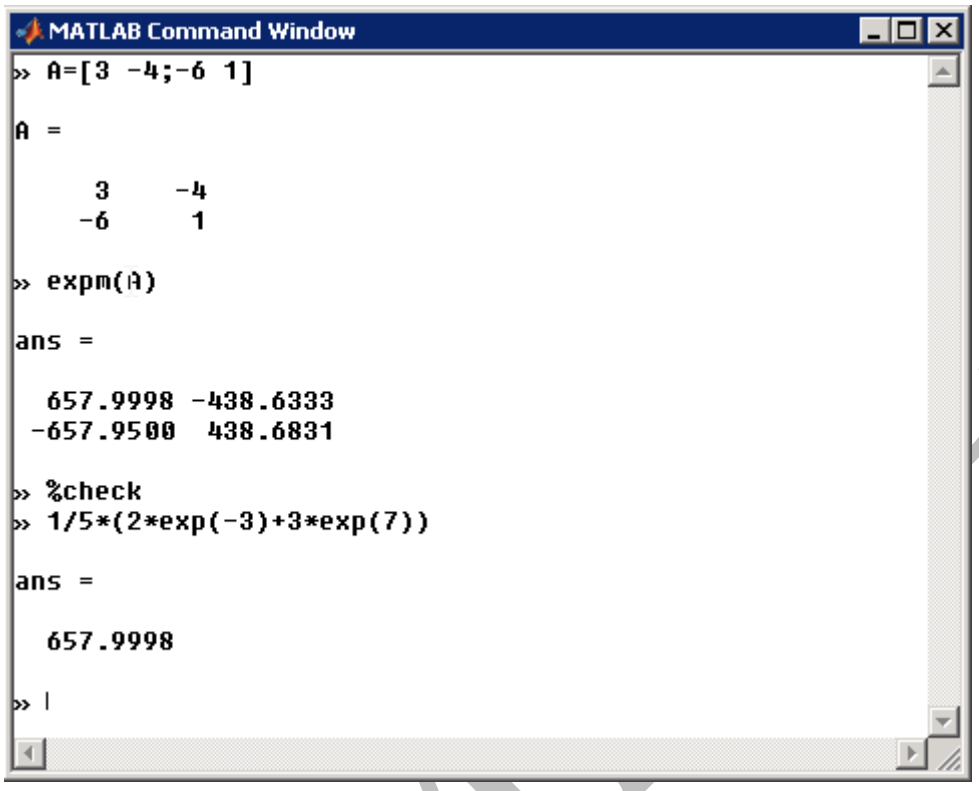
Now have a look the correlation matrix we have calculated by hand during our study of the principal component analysis proof and let us see if we obtain the same eigenvalues and eigenvectors by using the function **eig( )**:

SAMPLE



Everything is in conformity with what we calculated.

During the course on Integration and Differential Calculus we have seen that sometimes it is required to solved ODE by using matrix exponentiation. Let us have a look if MATLAB does the job correctly by taking the same example as in the theoretical course:

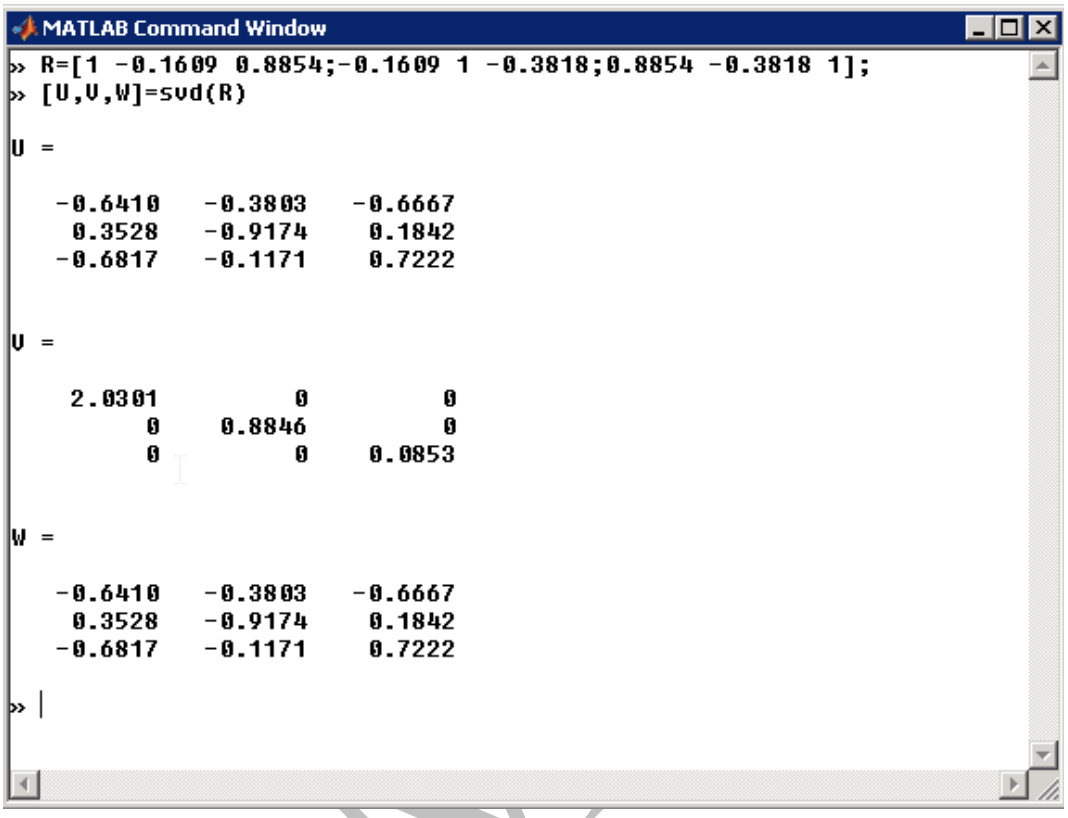


So we obtain the same results and can validate what MATLAB returns!

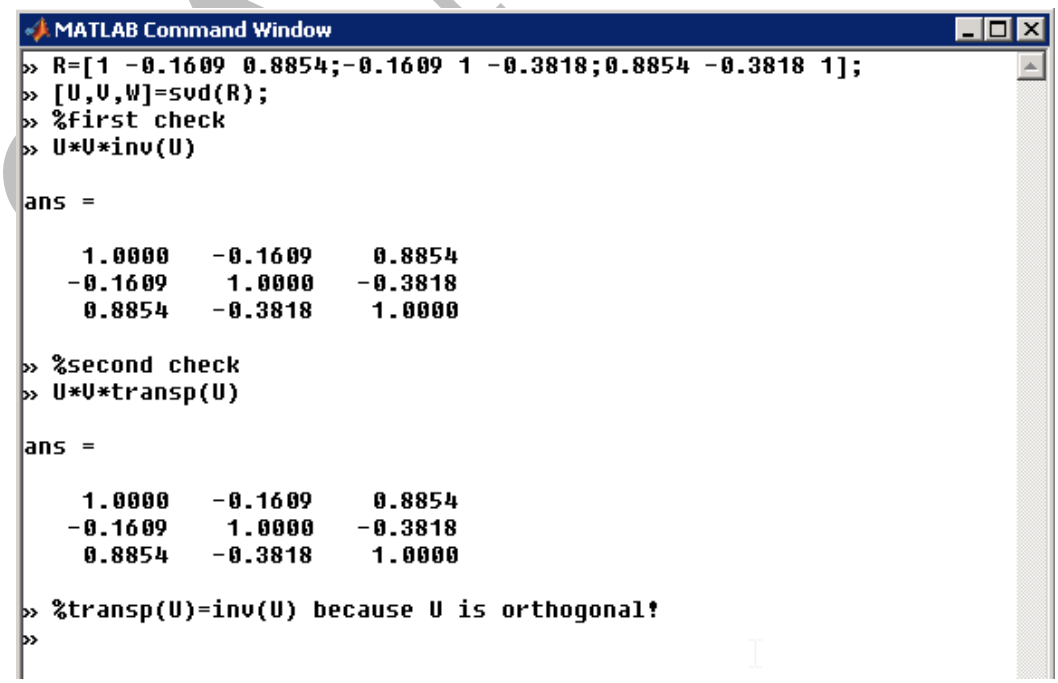
SAMPLE

In the training on Numerical Methods we have seen that the singular value decomposition has an important place in various statistical methods.

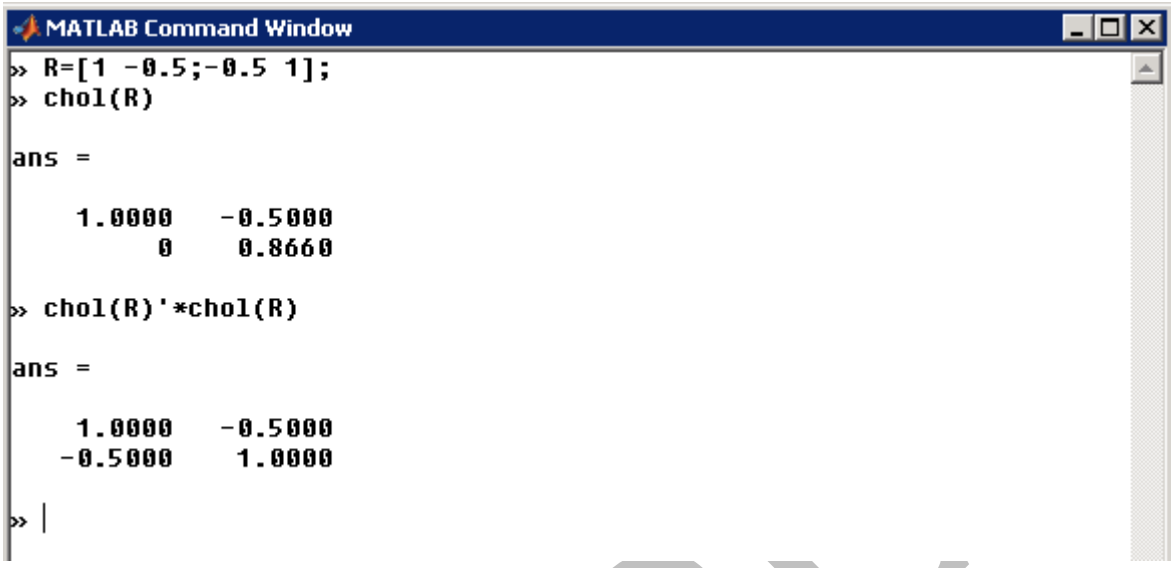
Using the correlation matrix calculated by hand in the theoretical training let us see if we obtain the same decomposition as calculated by hand by using the function **svd( )**:



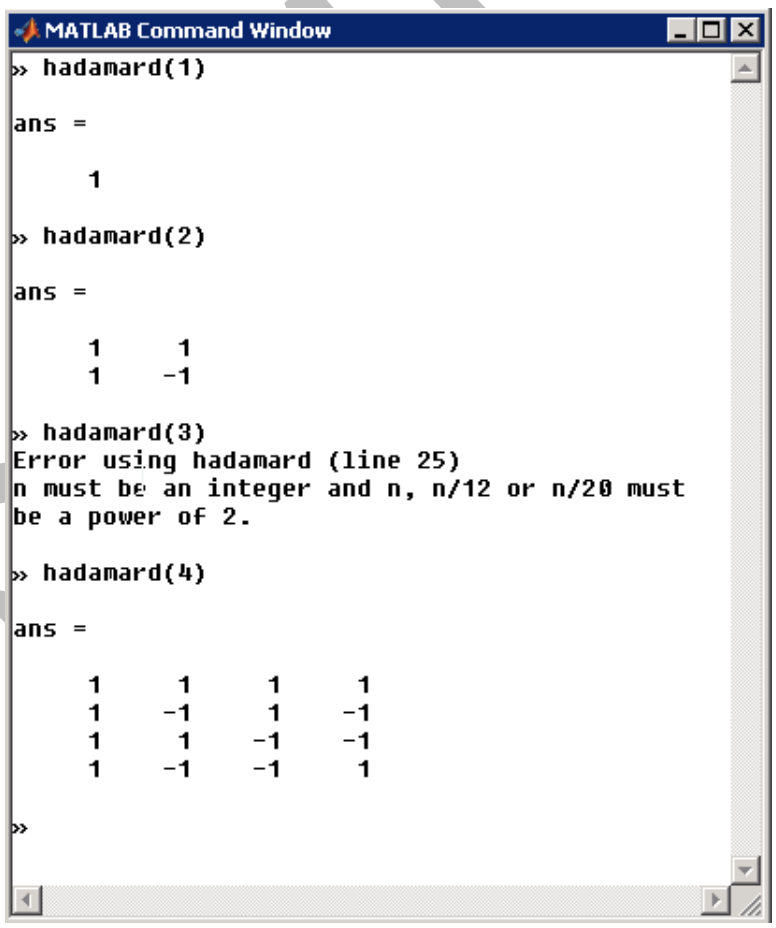
We can also see for fun if it is correct by inverting the calculation:



During our study of the bivariate normal distribution we have seen the importance of the Cholesky distribution. We can see if we obtain the same result as in the theoretical course by using the same correlation matrix and the using the function **chol( )**:



During the training on Design Of Experiment we have seen the importance of Hadamard matrices. Let us see if MATLAB respect the rules by using the function **hadamard( )**:



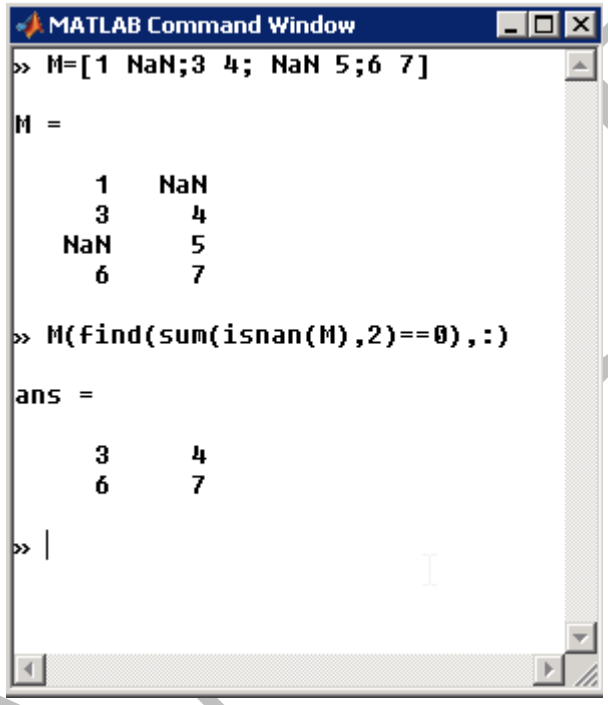
Everything is fine!

Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seems to have built-in functions.

Removing NaN in matrices (arrays)

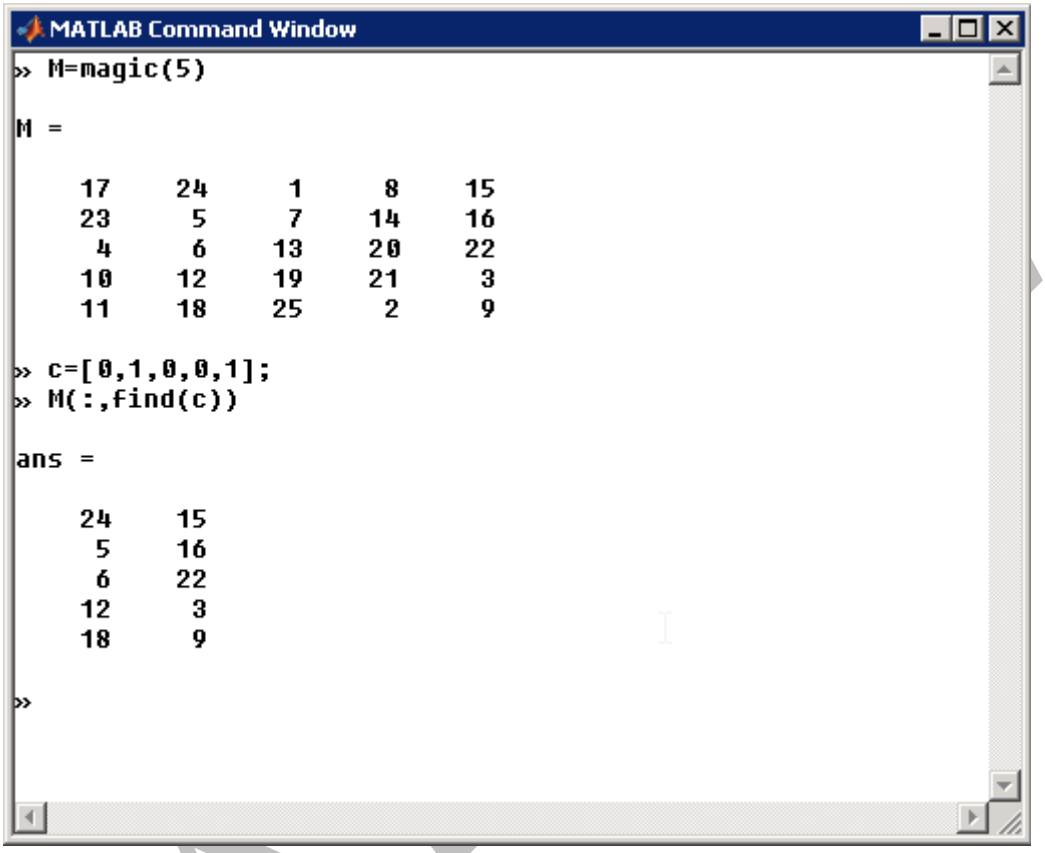
This is a typical request in finance but in the facts it could be better to interpolate using less or more advanced techniques the missing values (simple smoothing, or moving average, or spline, or...):



SAMPLE

Sampling matrix columns

In finance you have sometimes thousands of assets and you want to check your algorithms applied on multiples combinations of assets returns. Suppose we have such a matrix and that we have in our possession a vector (or a set of vectors) that makes the combinations unique (typically using Hadamard matrices). Then the trick is:



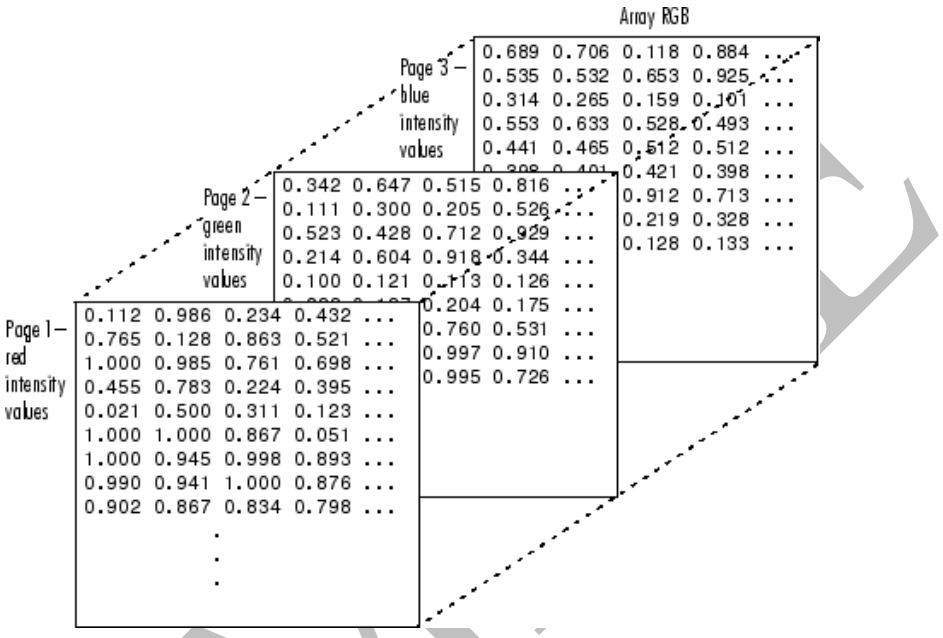
SA

Creation multiple dimensional matrices

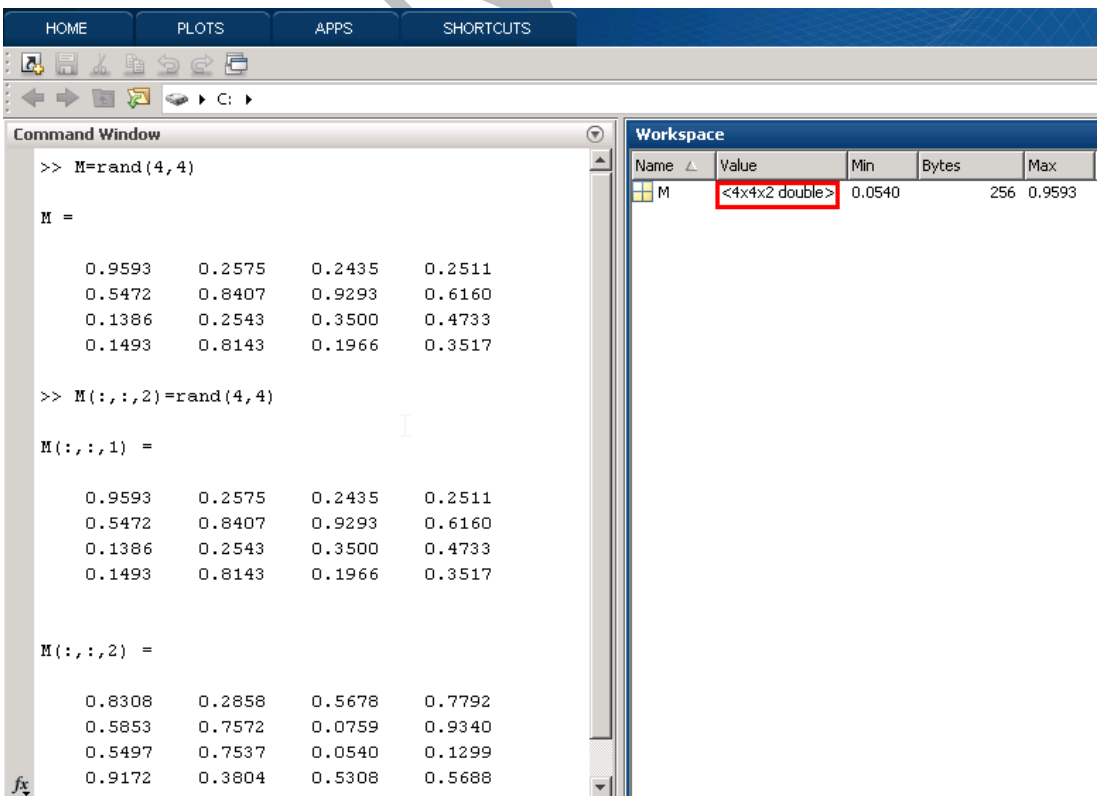
When you are working in MATLAB with images, by defaults they will be described by multiples dimensional array. This also sometimes happen with measures made by engineers.

Then it is important to know how it is structured and how to create such objects.

First here is an example on how an RGB image is structured in MATLAB:



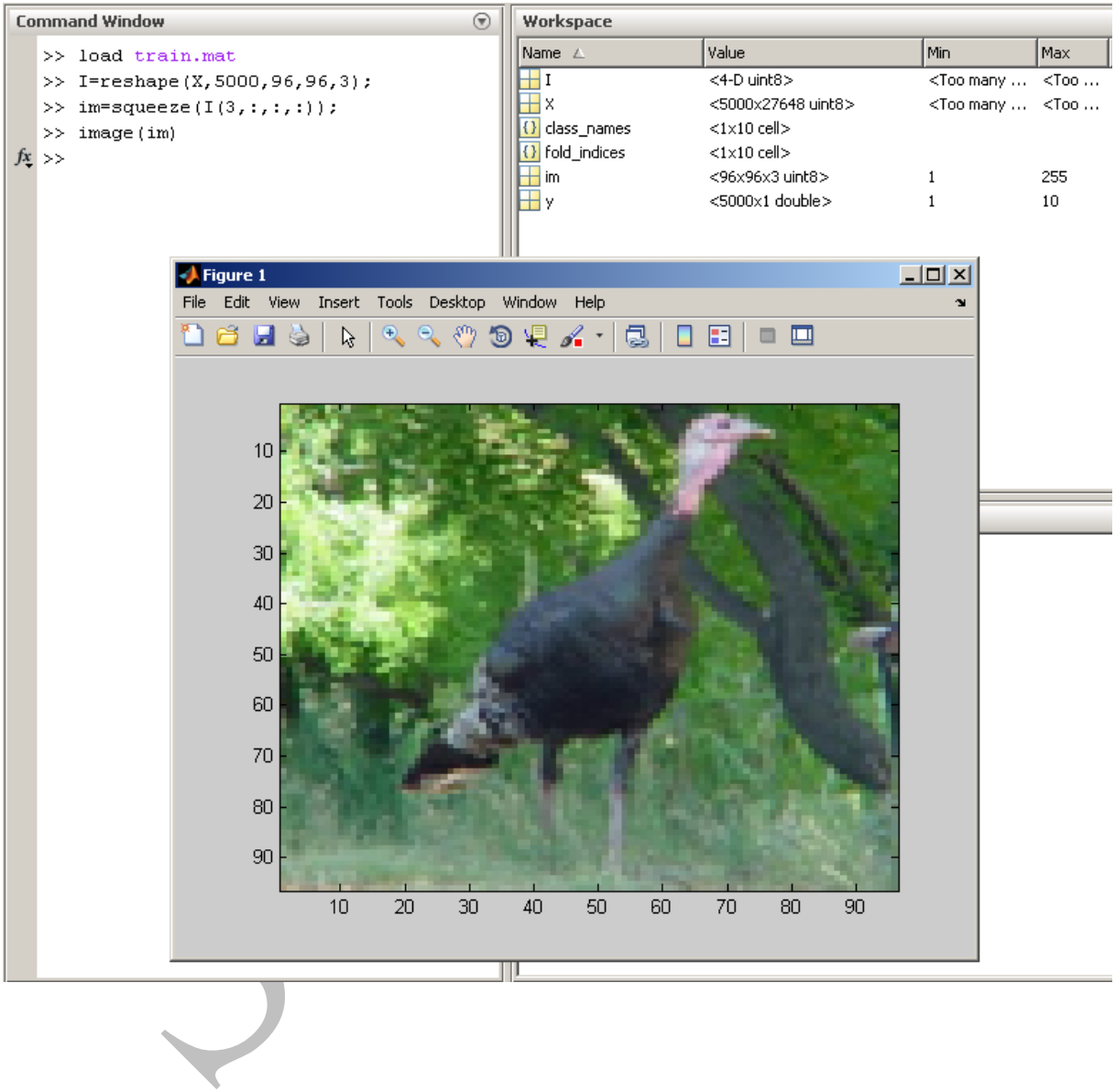
Now see how to build yourself such a multidimensional array:



We also can have 4 dimensional arrays! A well know example is the STL-10 deep learning image dataset of Stanford University course:

http://cs.stanford.edu/~acoates/stl10/

To load the *train.mat* file and show only one image we will use the **reshape( )** and **squeeze( )** functions as following to show the third of the 5,000 images:



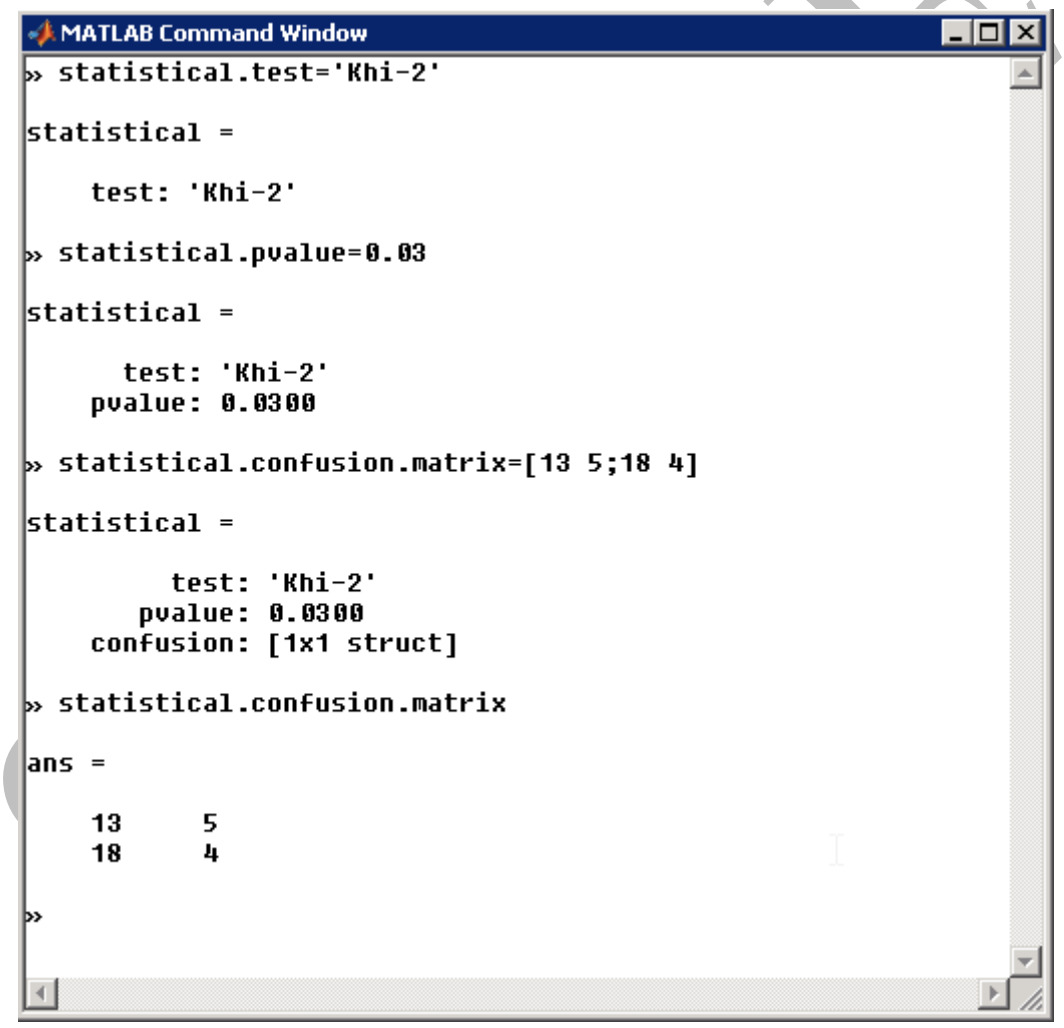
Manipulating tables (structure array)

You can since a few years work with tables structures in MATLAB like we do it for example in the R software. This is especially useful when you create your own scripts (see Scripting chapter on page 541).

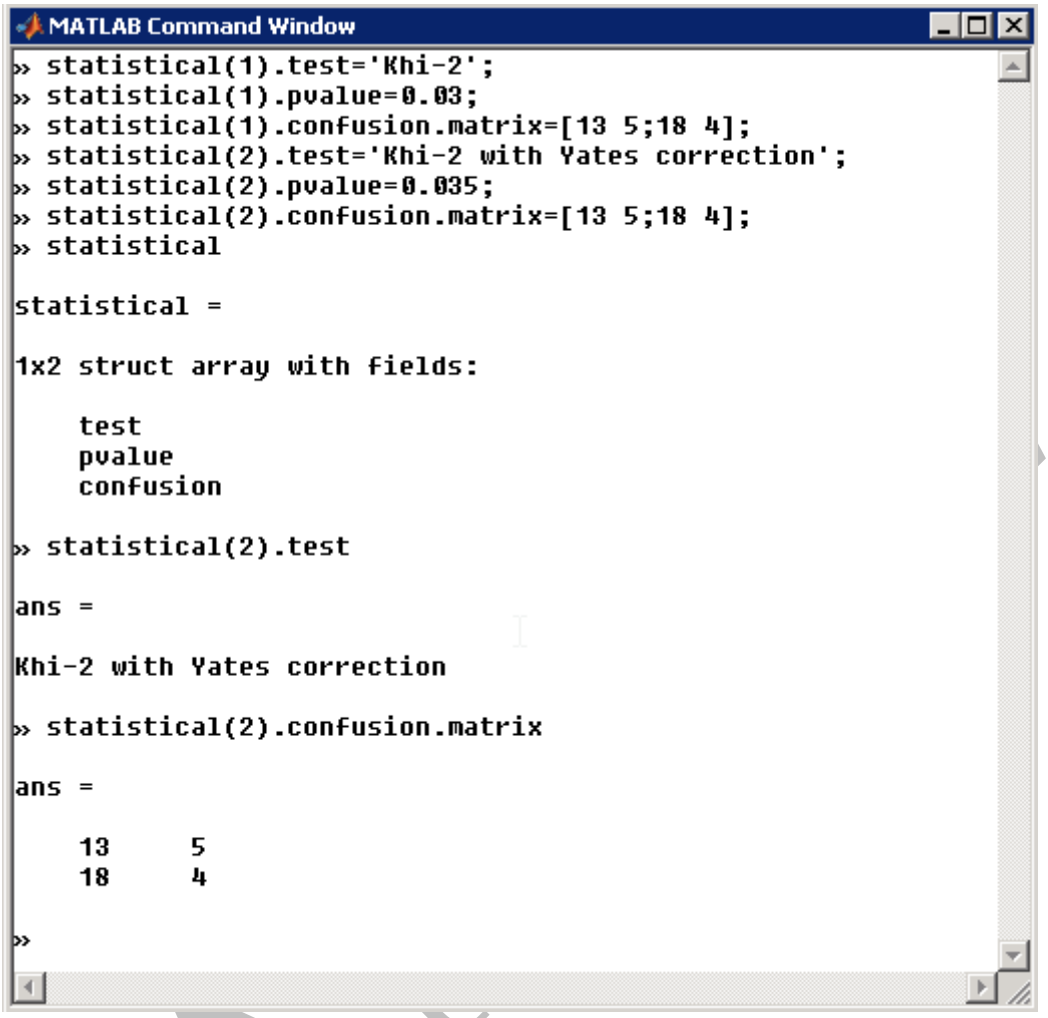
You have to know that some native MATLAB functions are by default structured arrays) and we already have meet such one when we have study the **mem** command.

Let us make a typical example well knows in R.

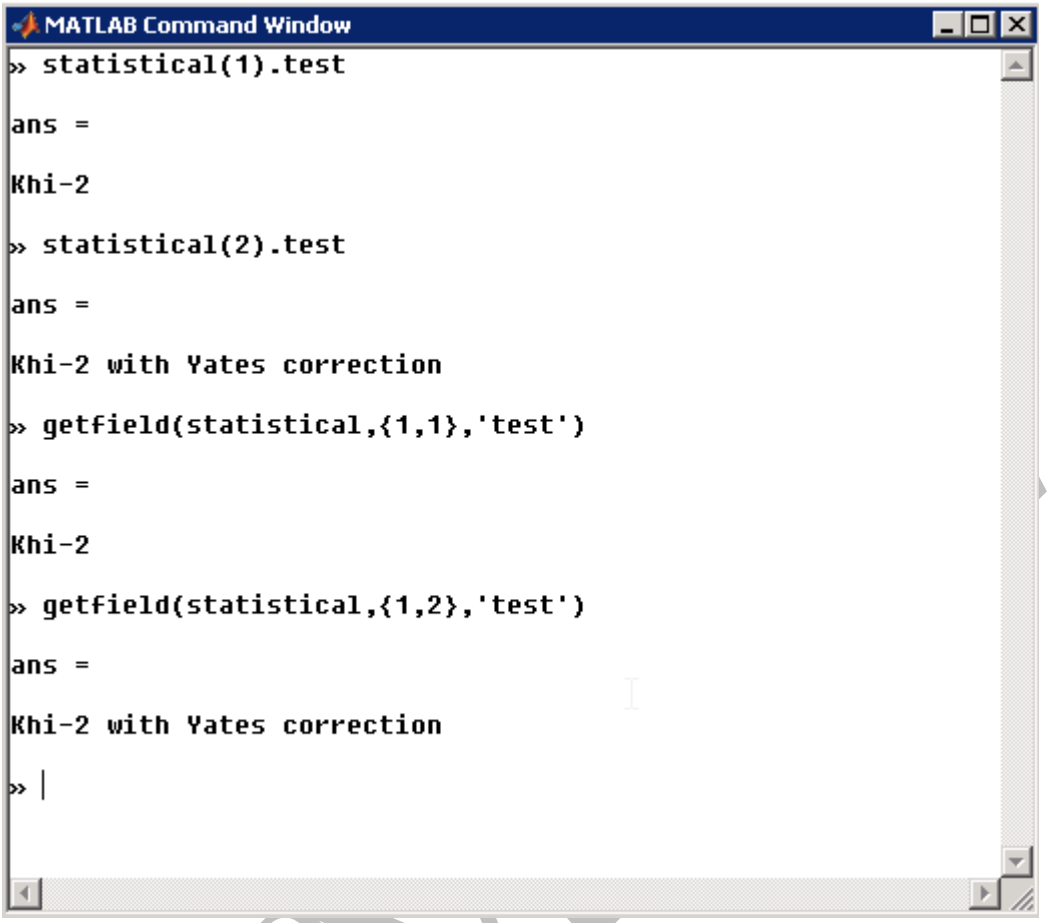
Imagine we have built a statistical function that returns a structure named *statistical* with different properties containing some values:



Now let us see how to access to the different properties values and to different property values:

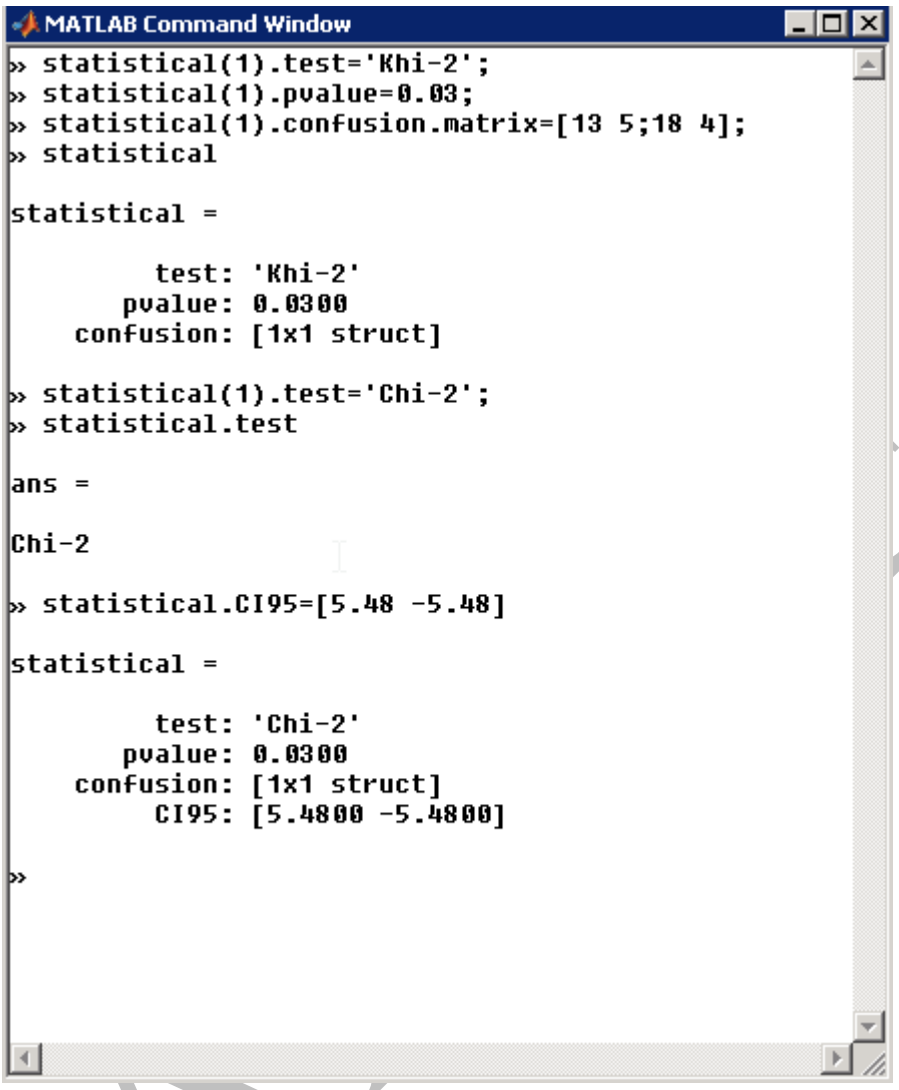


You must also know that there is another way to access to properties values:



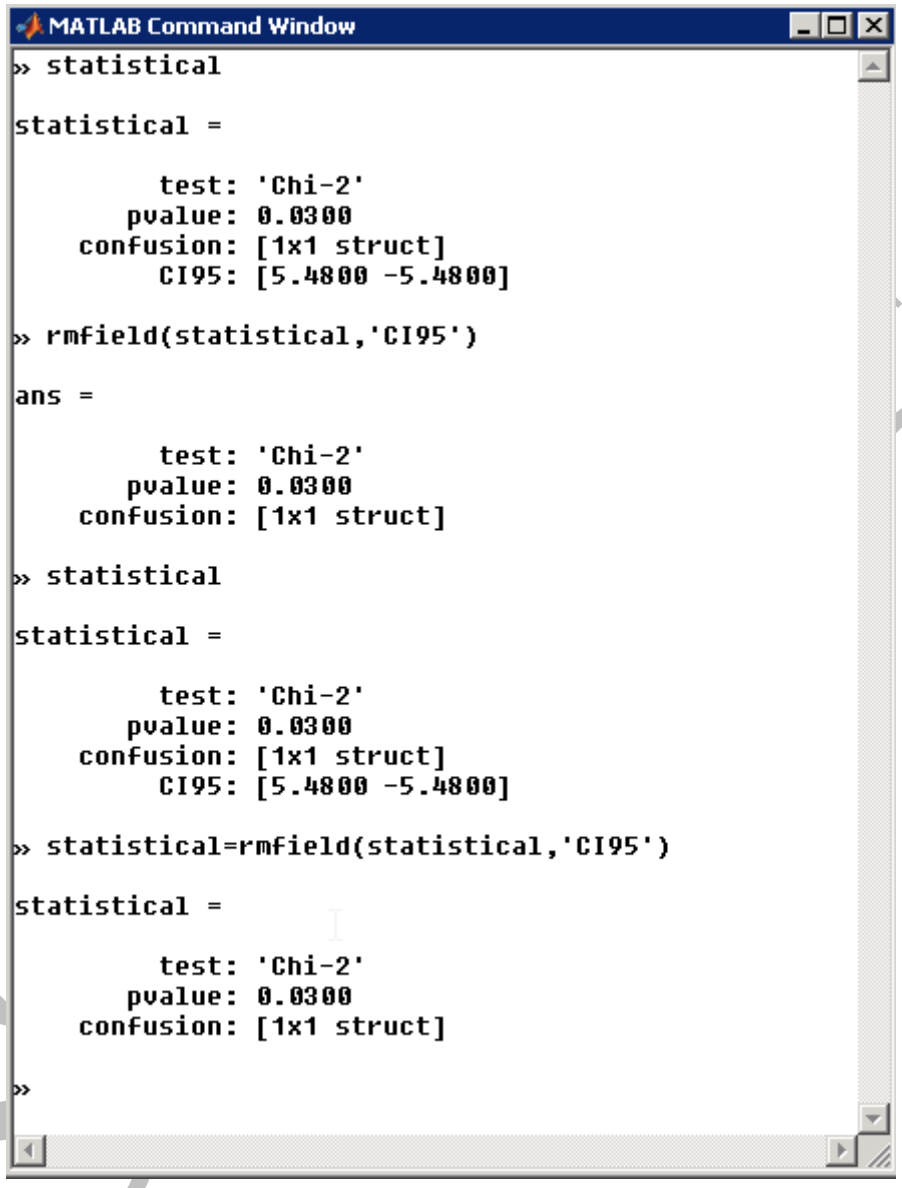
SAM

We can at any time change the value of each of the properties:

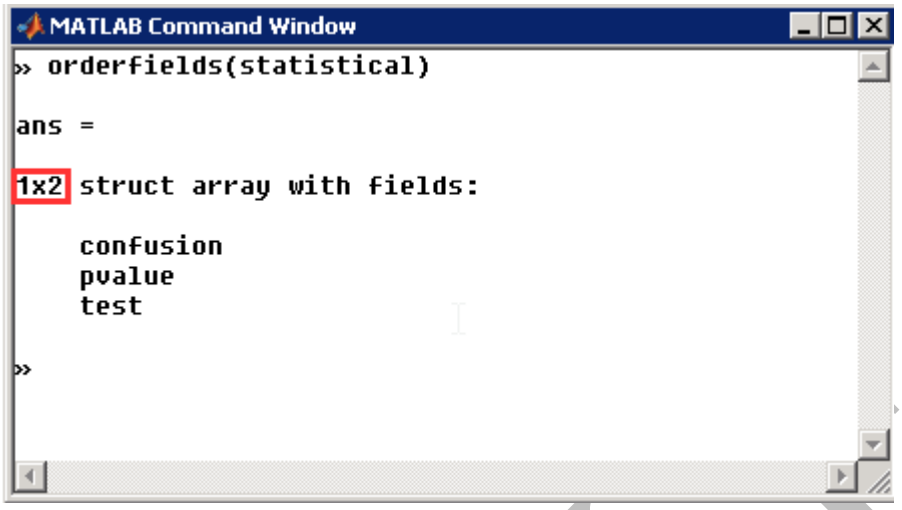


If you write the name of the structure you will by default get only the first dimension properties.

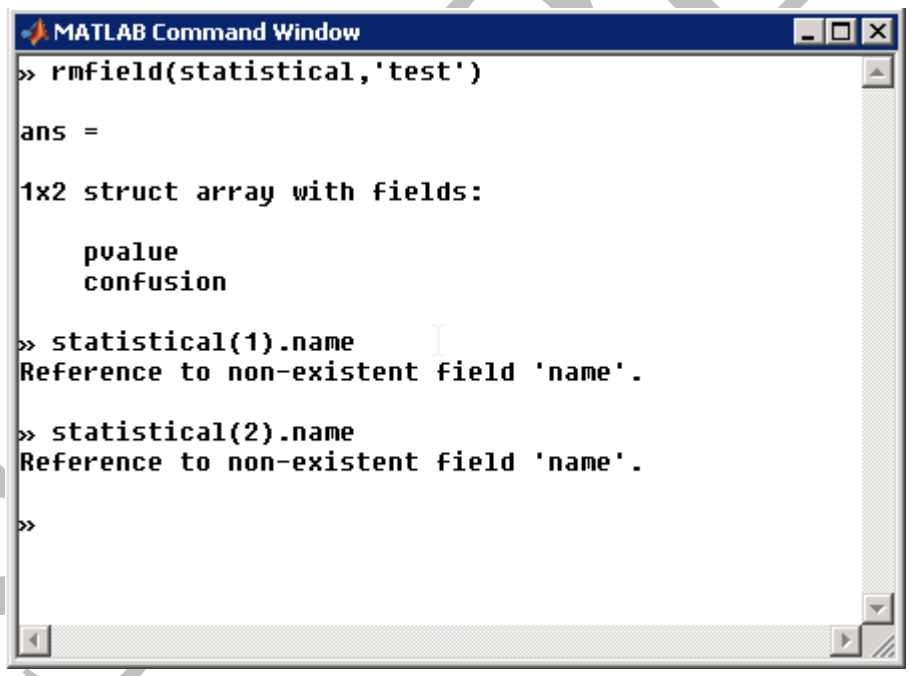
Using the function **rmfield( )** we can temporarily hide a property when we call a structured array:



We can get the list of the names of the structured array using the function **orderfields( )** and at the same time we get the number of dimensions:



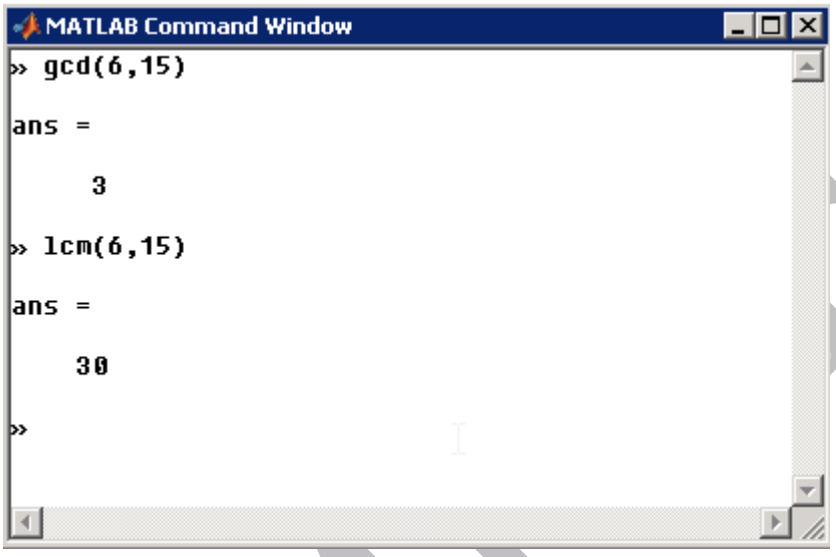
Strange behavior... if we remove a field containing a string this will be definitively removed at the contrary of a property containing a matrix:



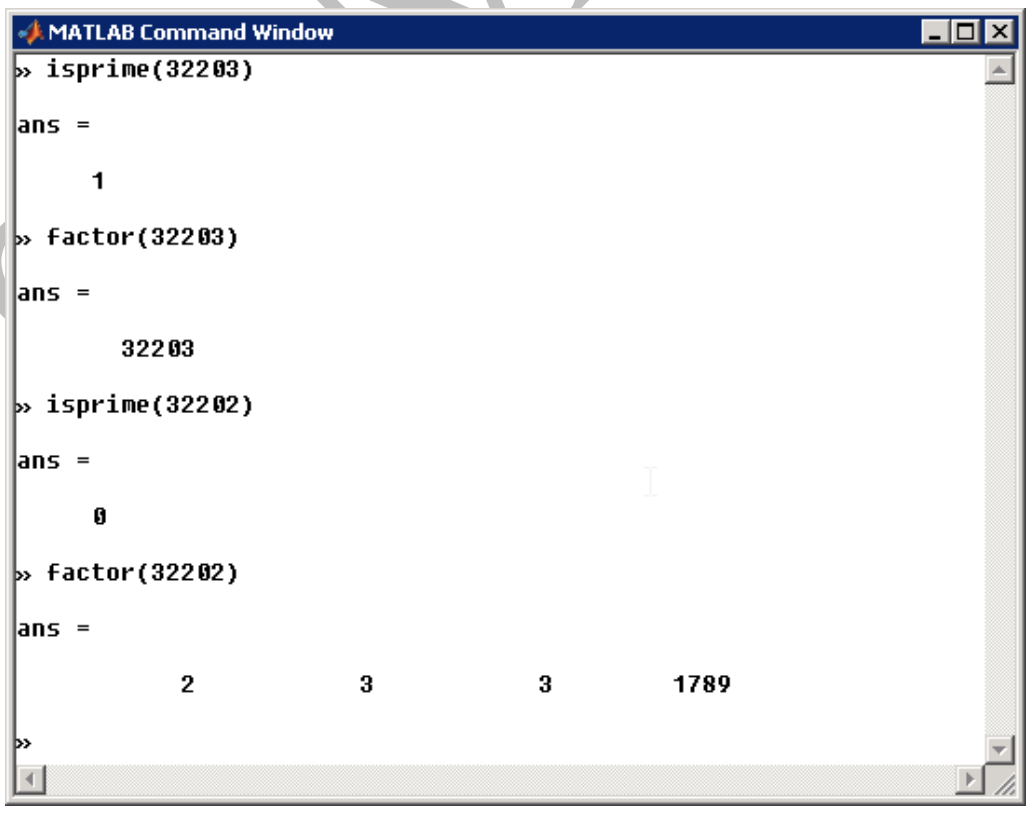
Mathematical functions

The purpose now is to check that MATLAB has some of the mathematical function that we used we saw the Applied Mathematics e-book.

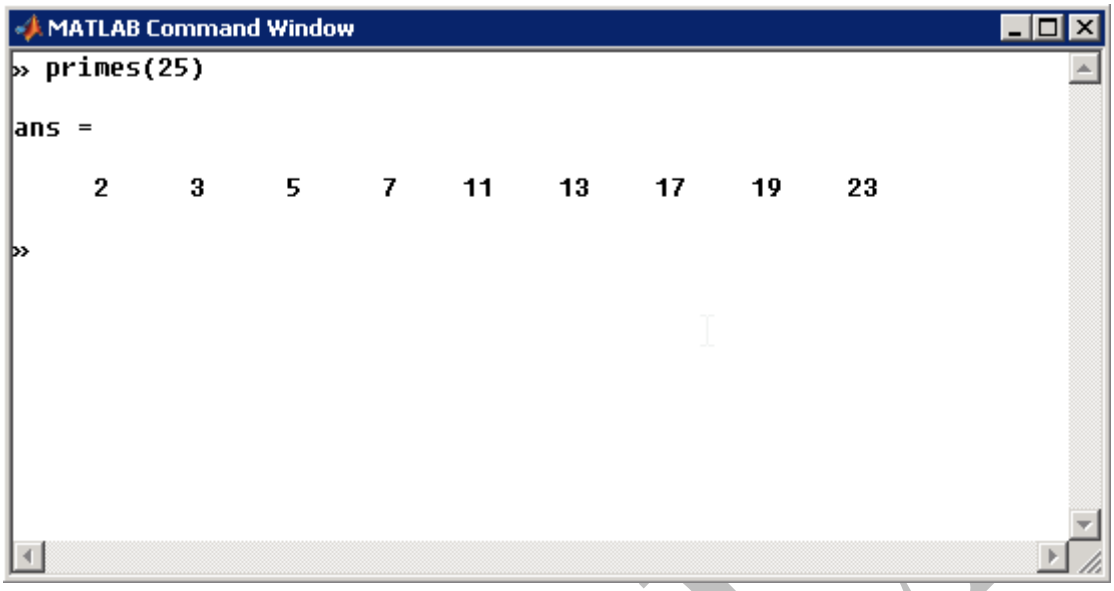
Let us start by the greatest common divisor with the function **gcd( )** and also the lower common multiple with the function **lcm( )**:



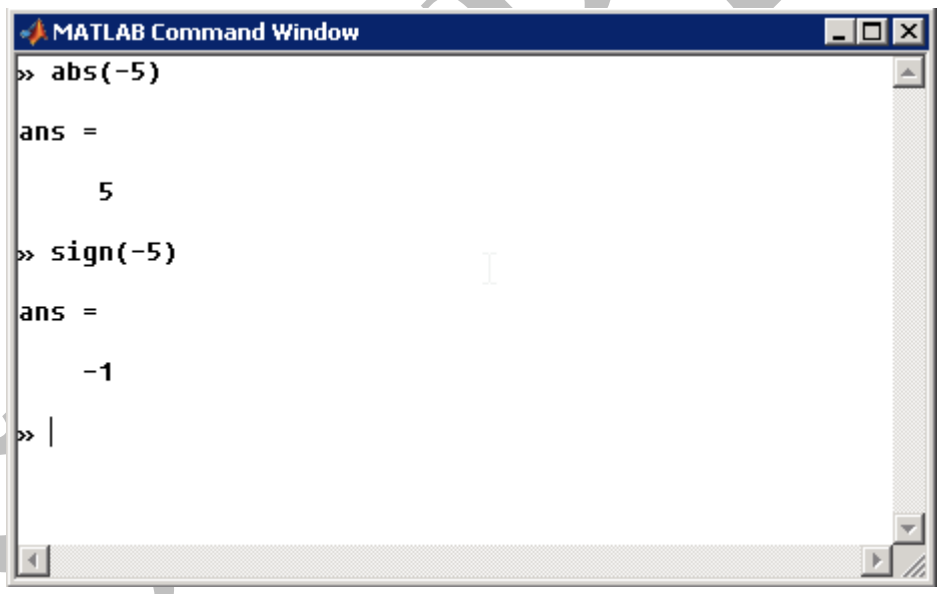
We also know that in R.S.A. cryptography that the manipulation of prime number is important. For this purpose MATLAB has some prime relative functions like **isprime( )**, and **factor( )**:



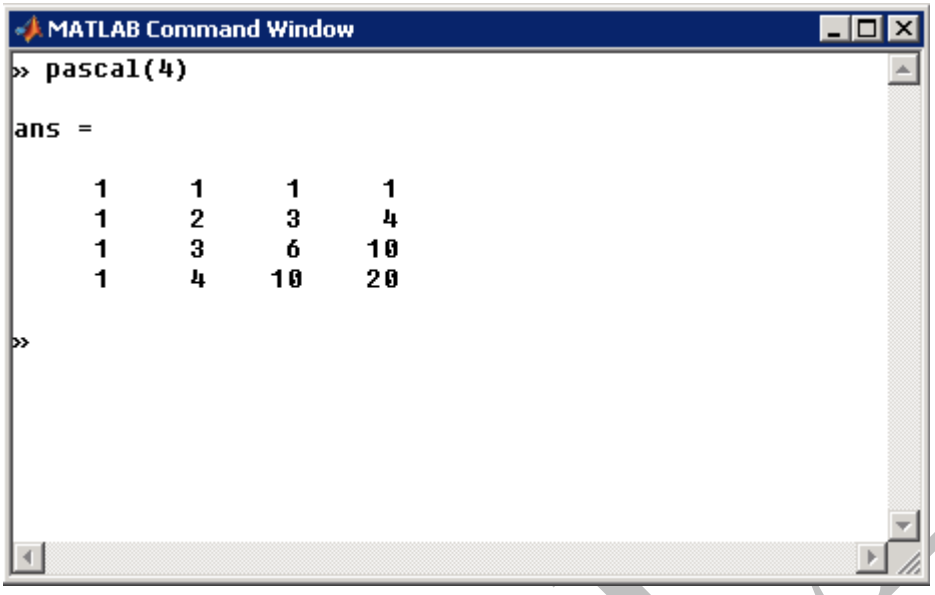
and also **primes( )**:



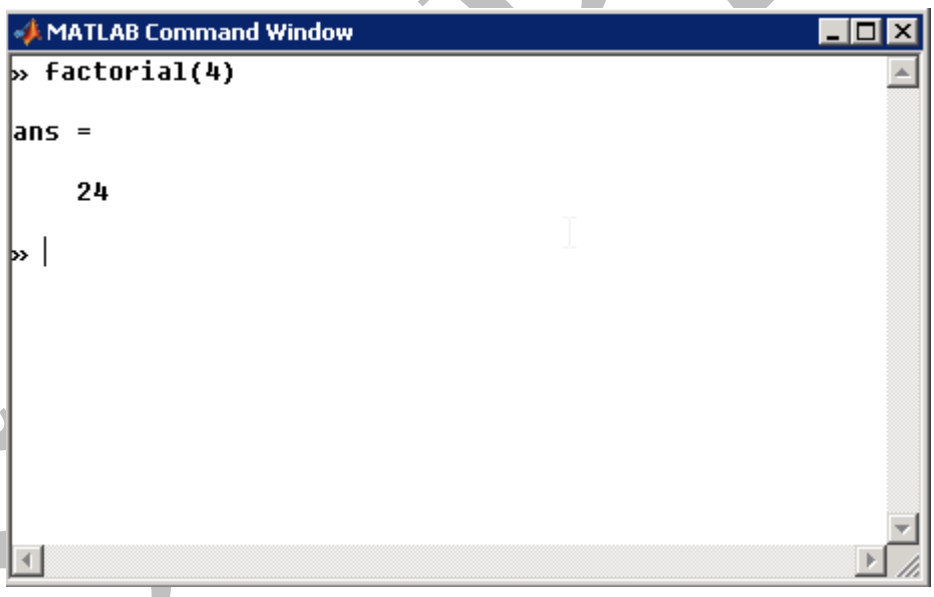
For sure we have the absolute value that we have already seen twice before with the function **abs( )** and the signum function **sign( )** that is still useful in numerous applied field:



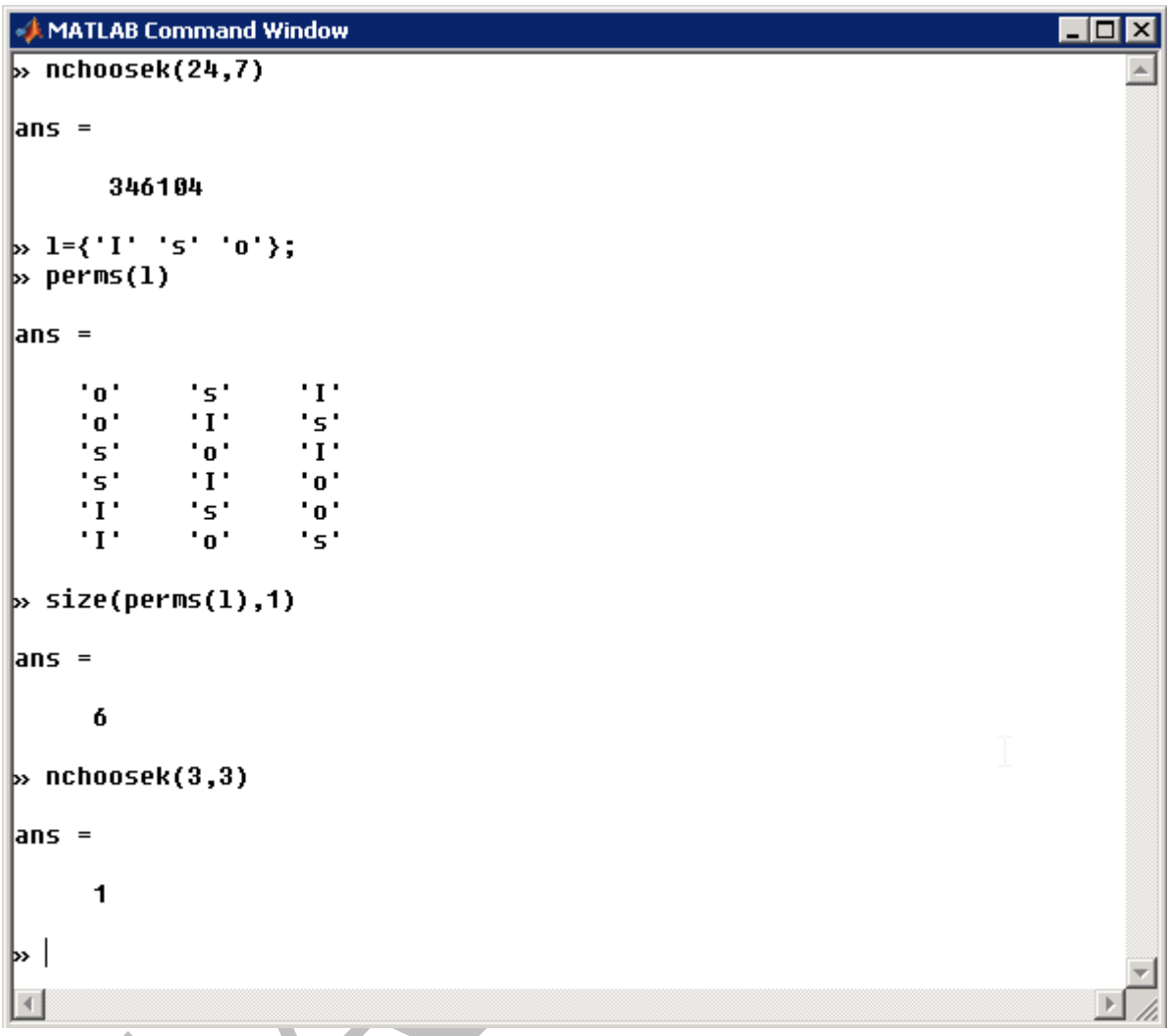
For high school students you can get the Pascal triangle with the function **pascal( )** for fun...:



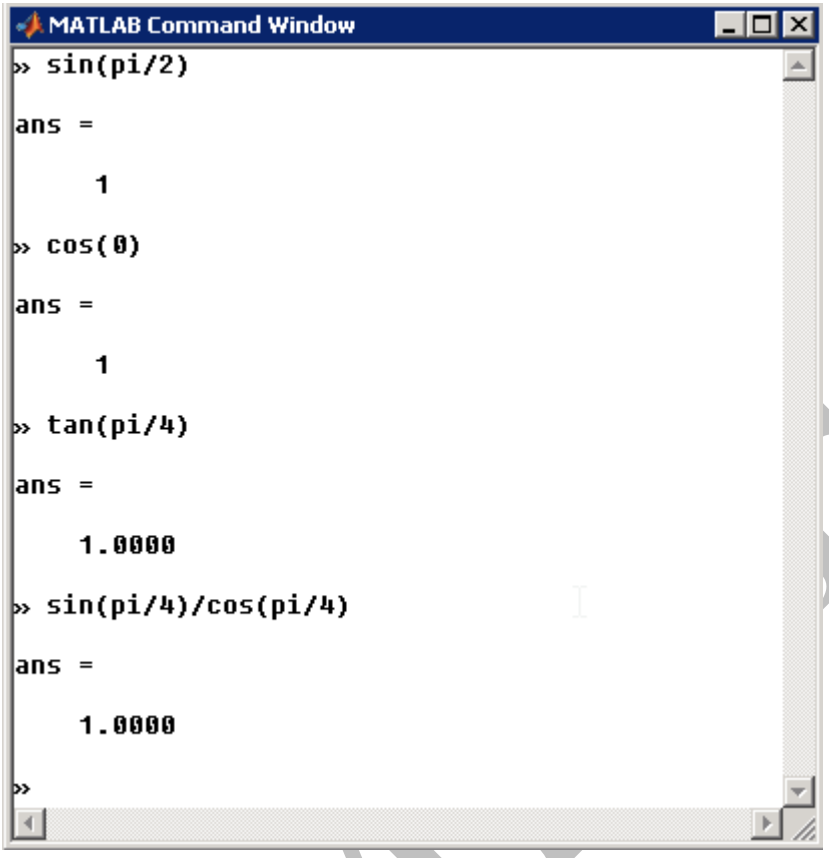
And our tradition **factorial( )** function very useful in statistics, number theory and quantum physics:



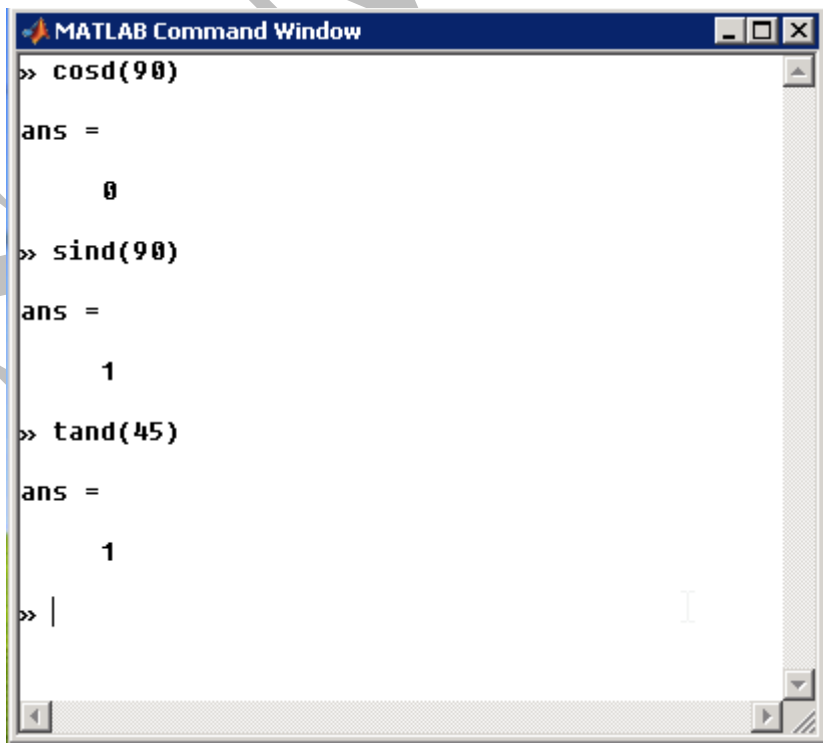
Another important function in statistics is the combinatorics that you can get with **nchoosek( )** and for high school students they can play with also with the **perms( )** function to see all possible permutations using the sample examples as in the theoretical course:



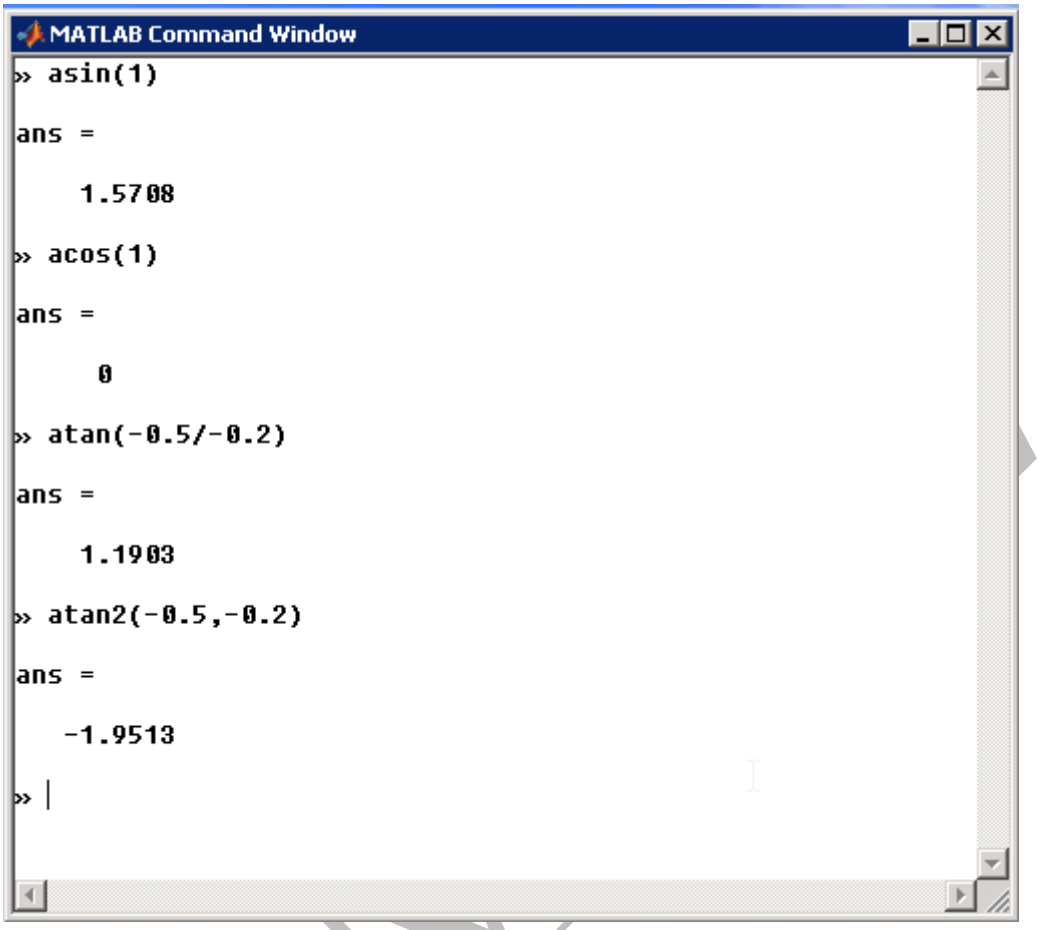
Now some trigonometry based on radians with the functions **sin( )**, **cos( )**, **tan( )**:



For people that like to work in degrees instead of radians you have the functions **cosd( )**, **sind( )** and **tand( )** that are useful when you work with latitudes and longitudes:

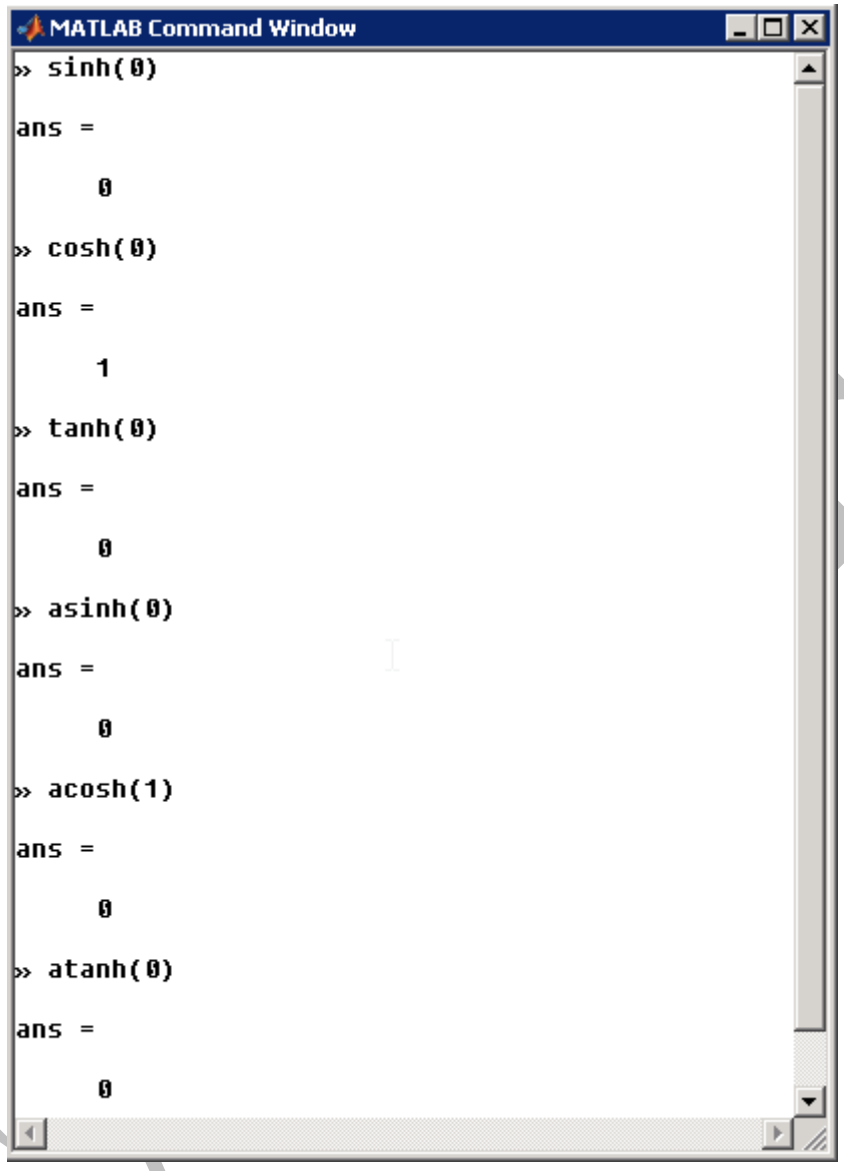


let us continue in radians with some inverse trigonometric functions in radians like **asin( )**, **acos( )** or **atan( )** or the important **atan2( )**:

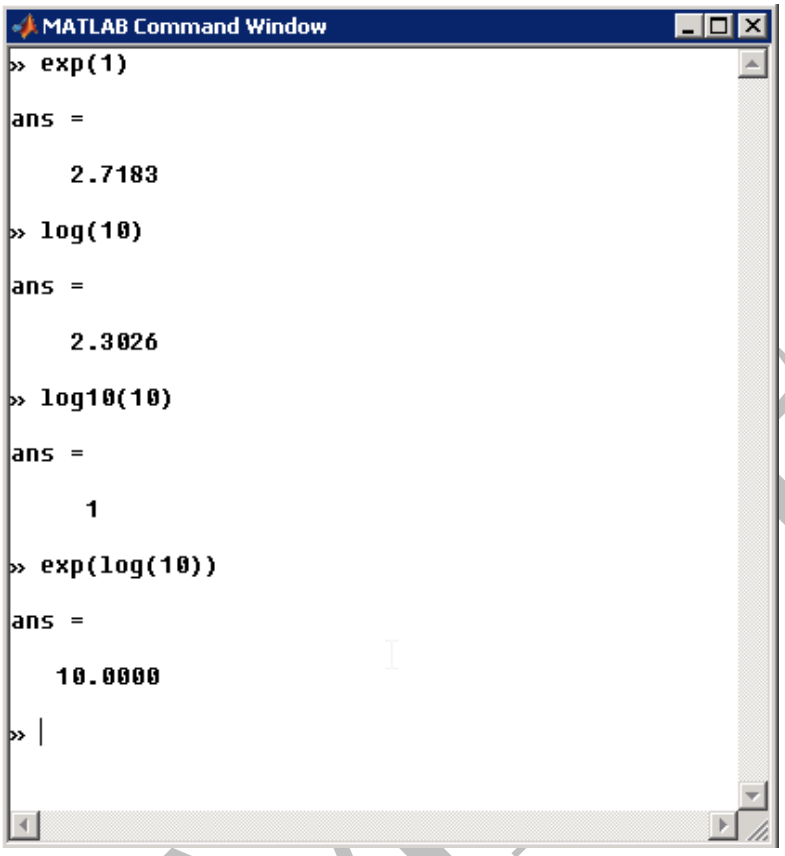


SAM

And for sure we also have the hyperbolic normal and inverse trigonometric functions **sinh( )**, **cosh( )**, **tanh( )**, **asinh( )**, **acosh( )**, **atanh( )**:

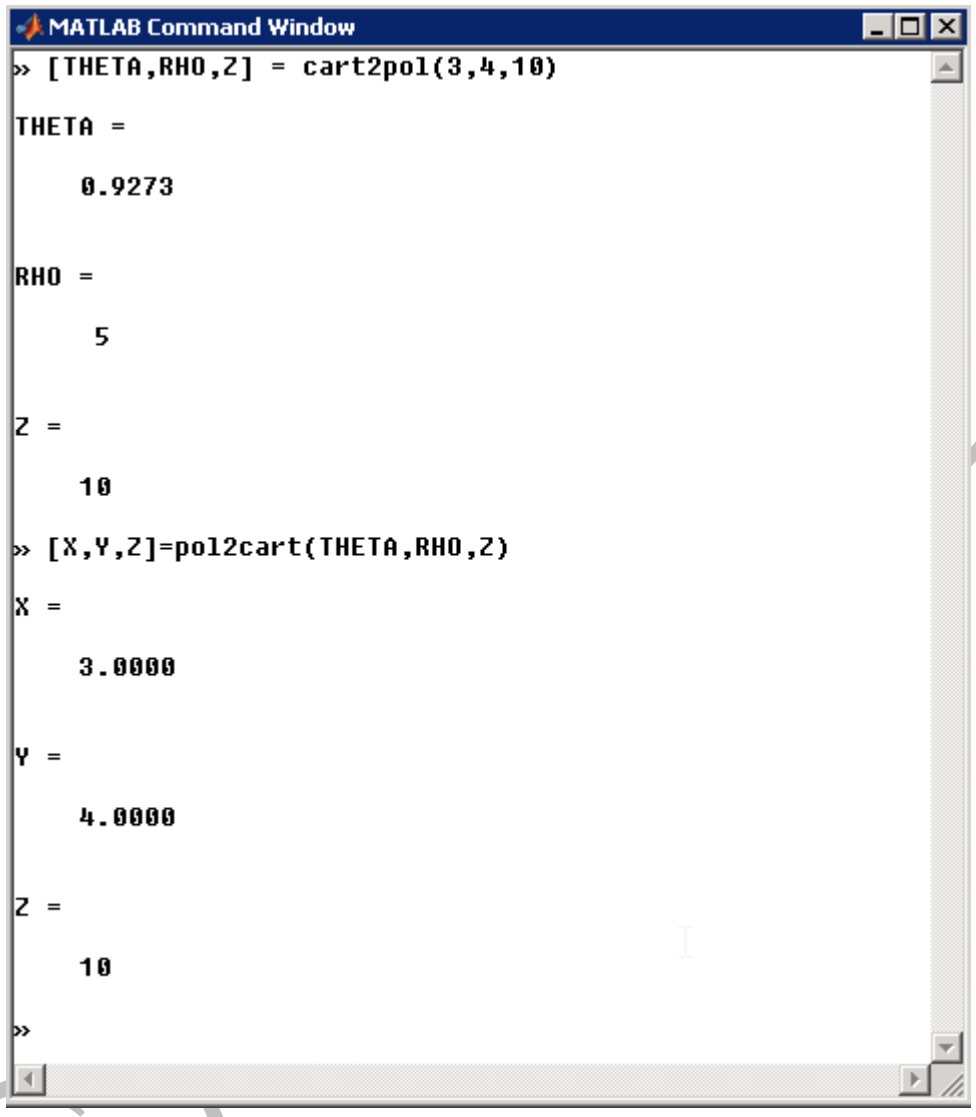


Some other very classic and useful mathematical functions are exp( ) for the exponential, **log( )** for the natural logarithm, **log10( )** for the logarithm in base 10:

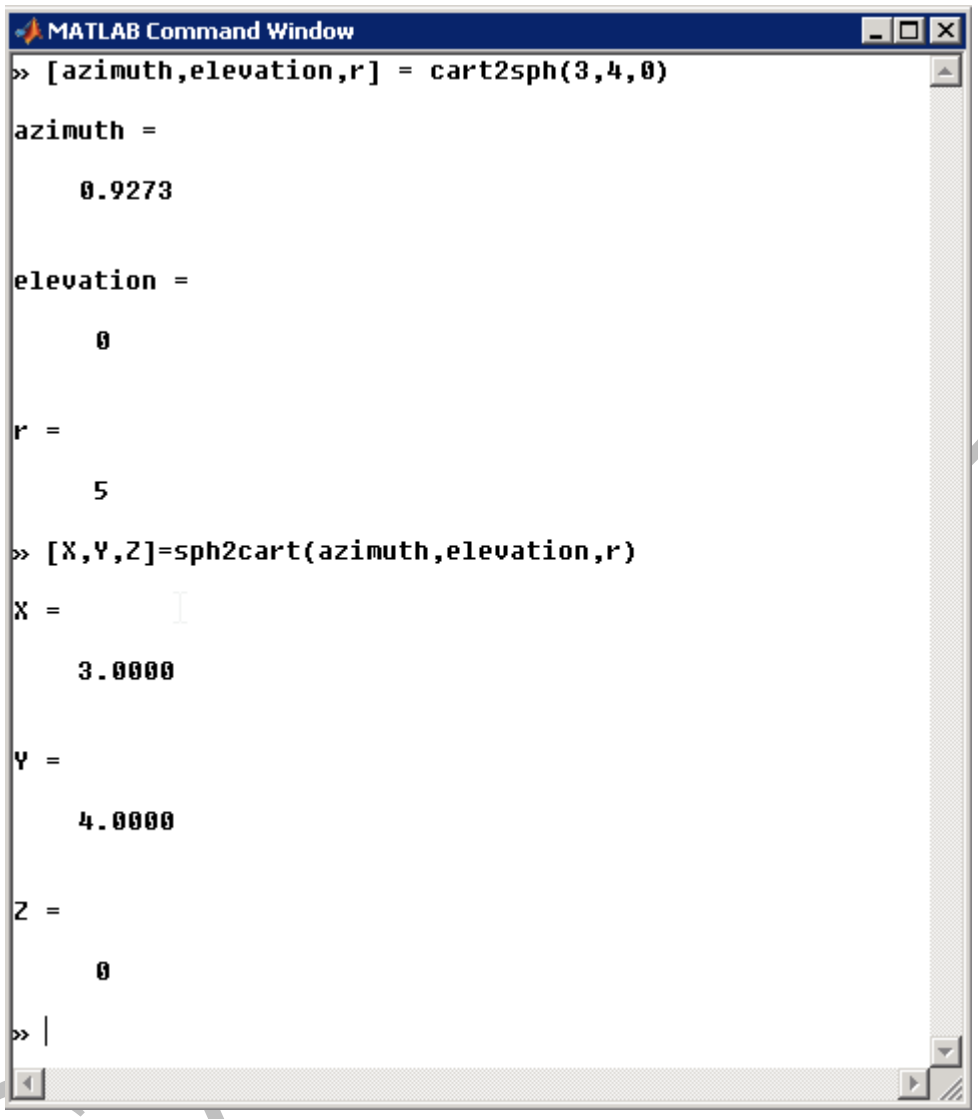


SAMPLE

Sometimes it is useful to some charts to know how to transform Cartesian coordinates into polar coordinates. This can easily be done thanks to the function **cart2pol( )** or also the inverse by using the function **pol2cart( )**:



In astronomy, or celestial mechanics and also in atomistic in can be useful to know how to convert Cartesian coordinates to spherical coordinates using the function **cart2sph( )** or the inverse operation using **sph2cart( )**:



Now let us see a famous example using complex number and trigonometric function by plotting one of the 3*d* orbital of the hydrogenoid atom:

$$Y_{2,2} = \sqrt{\frac{15}{32\pi}} \sin^2\theta e^{2i\phi}$$

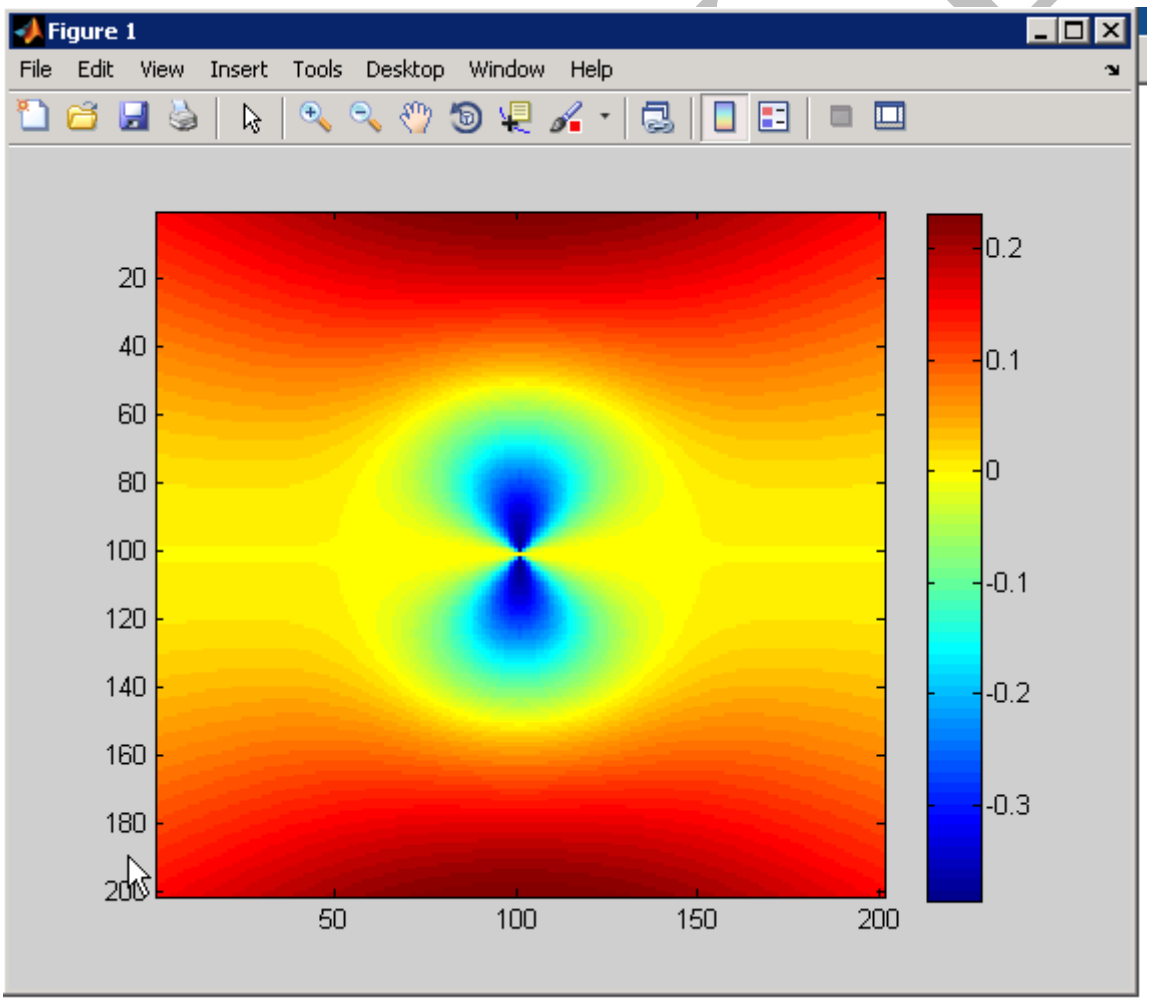
```
%probability densities
pd_psi_210 = @(r,theta,phi) real(exp(2*i*phi)).*(sin(theta).^2)*sqrt(15/(32*pi));

%configuring the range
[x, y, z] = meshgrid(-1:.01:1,-1:.01:1,-0.5:.01:0.5);
[theta,phi,r] = cart2sph(x,y,z);

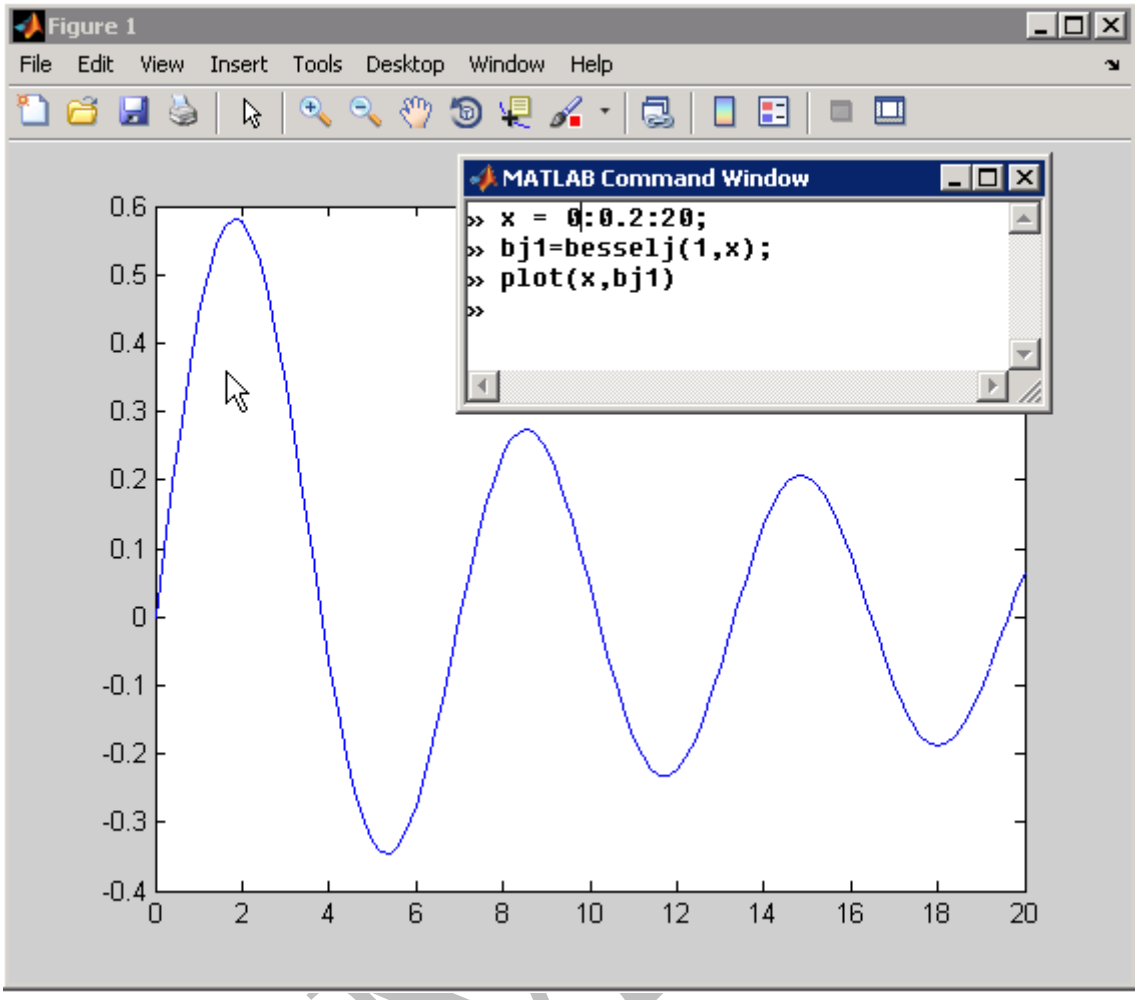
%create array of probability density magnitudes
psi_210_vals = pd_psi_210(r,theta,phi);

%plotting
imagesc(psi_210_vals(:,:,1)); %x-y plane
colorbar
```

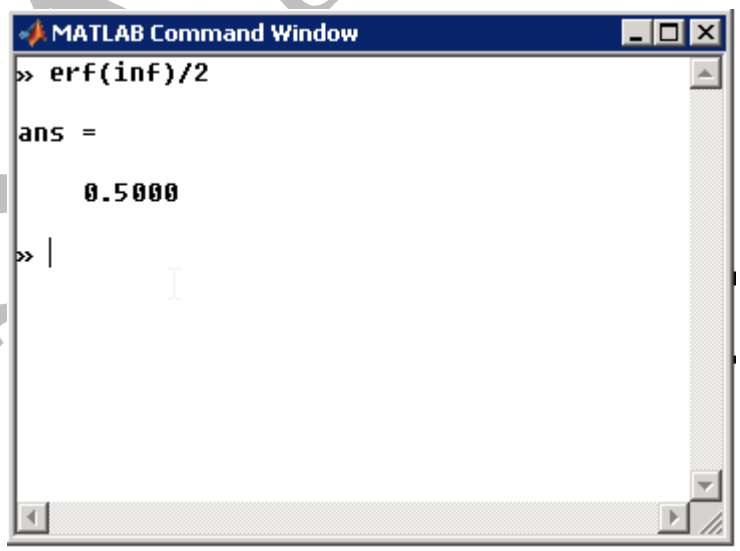
That gives:



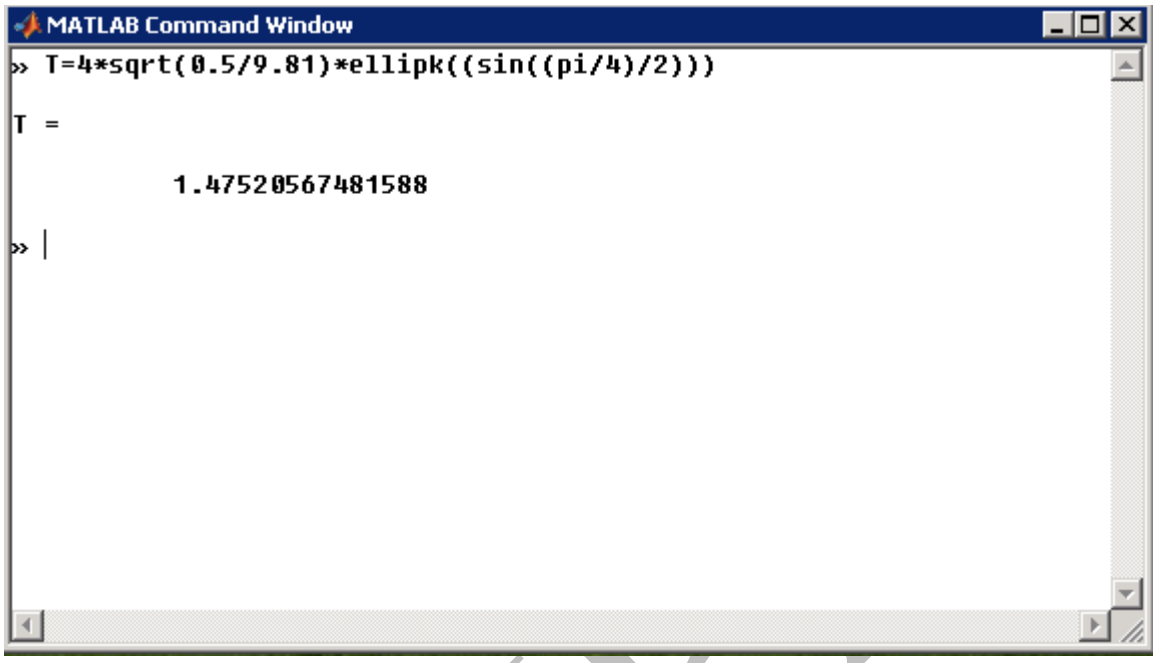
We have seen in the Optic course and Nuclear course that the Bessel J function was very useful. Late us take a look if it's available in MATLAB using the function **besselj( )**:



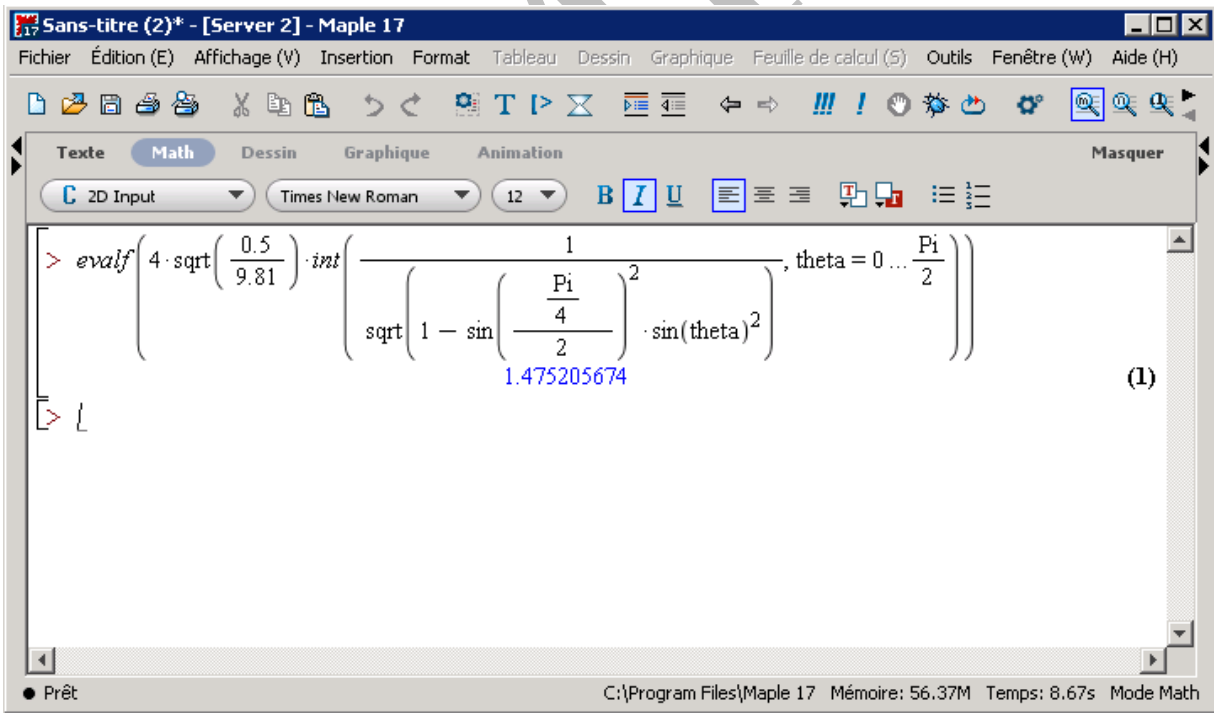
We have also see in the Statistic course and Thermodynamics course a function named "erf". Let us see if this function is really available in MATLAB using the **erf( )** function:



During our study of nuclear magnetic resonance bird cage we have seen that K elliptic functions were useful. Let us have a look with a typical academic example base on the pendulum using the function **ellipk( )**:



Is the result very accurate? Let us check that with Maple 17:

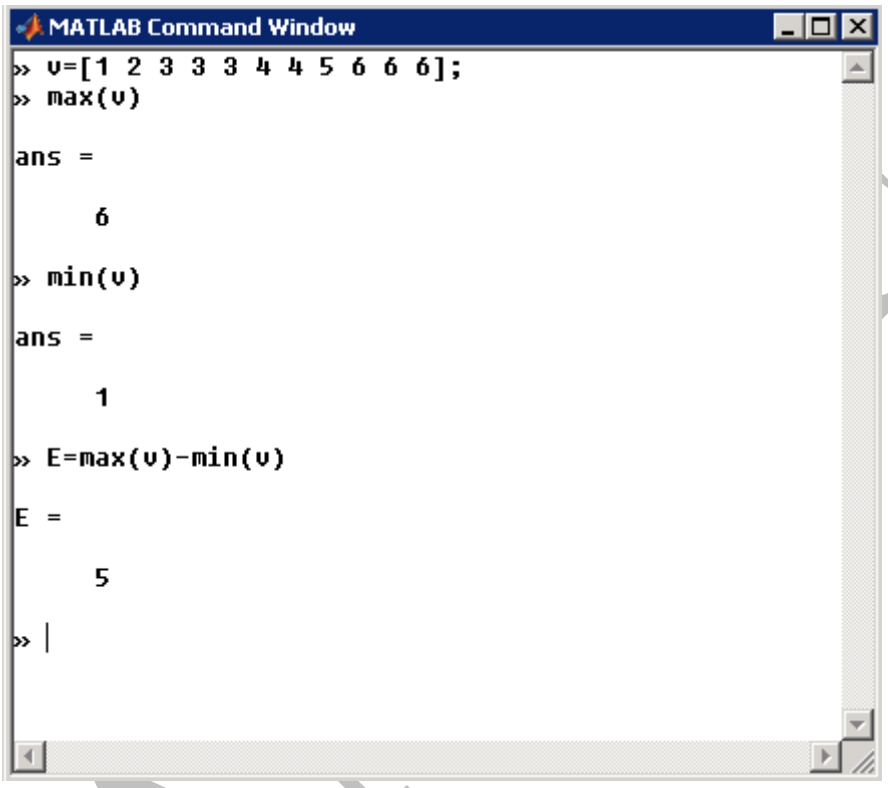


Statistical (core elementary) functions

MATLAB is very limited in native advanced statistical functions if you do not have additional Toolbox like the Statistical Toolbox, Econometrics Toolbox or Forecasting Toolbox.

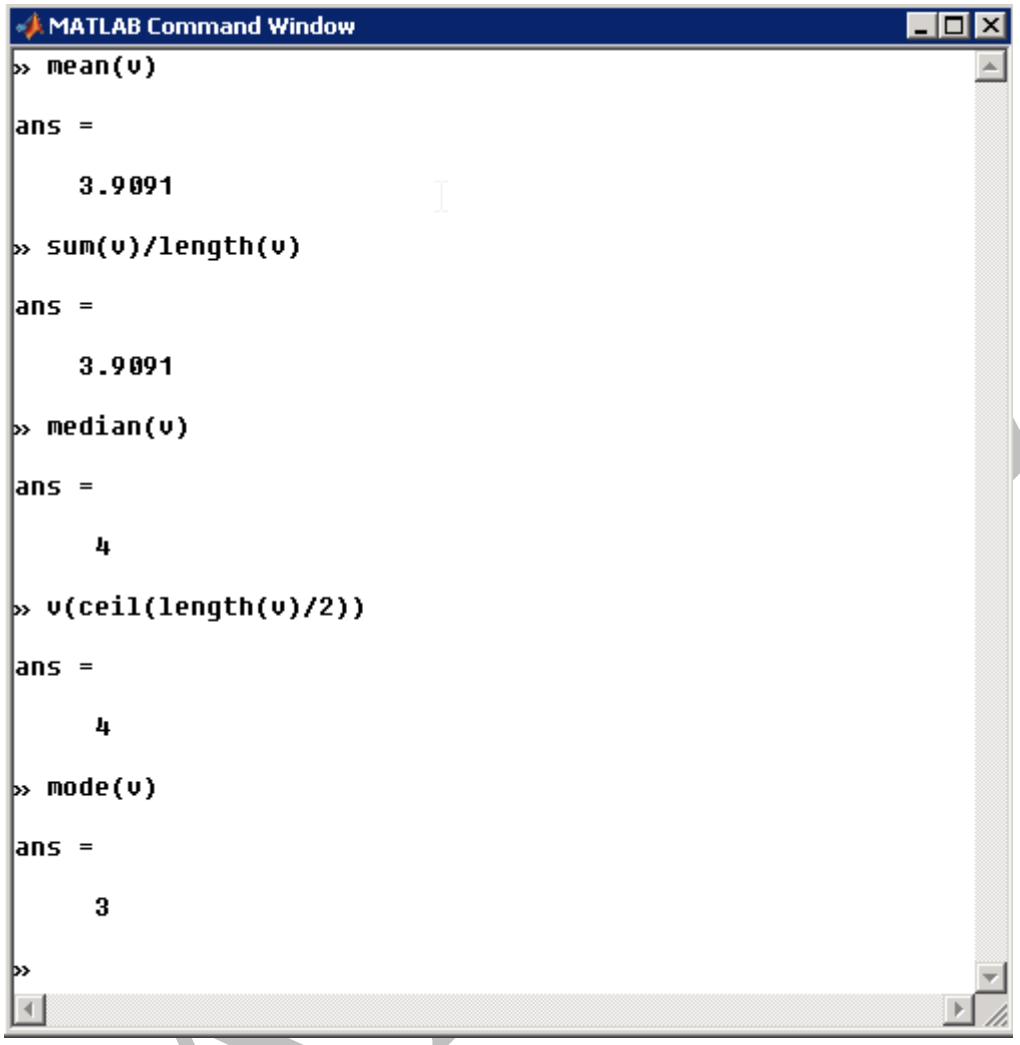
The purpose here is to see what is available in the MATLAB core version.

First take a look the **max( )**, **min( )** function and calculate also the range of a vector:

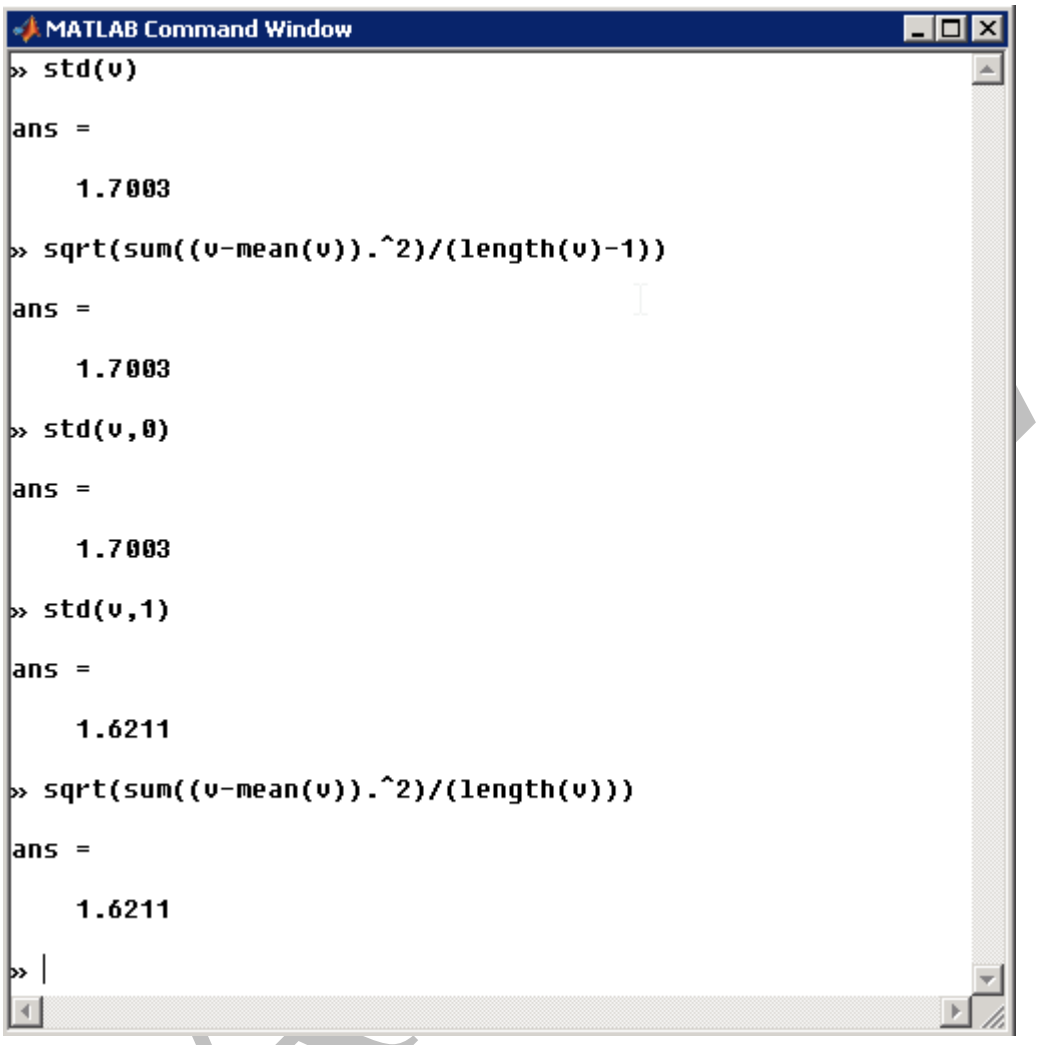


SAMPLE

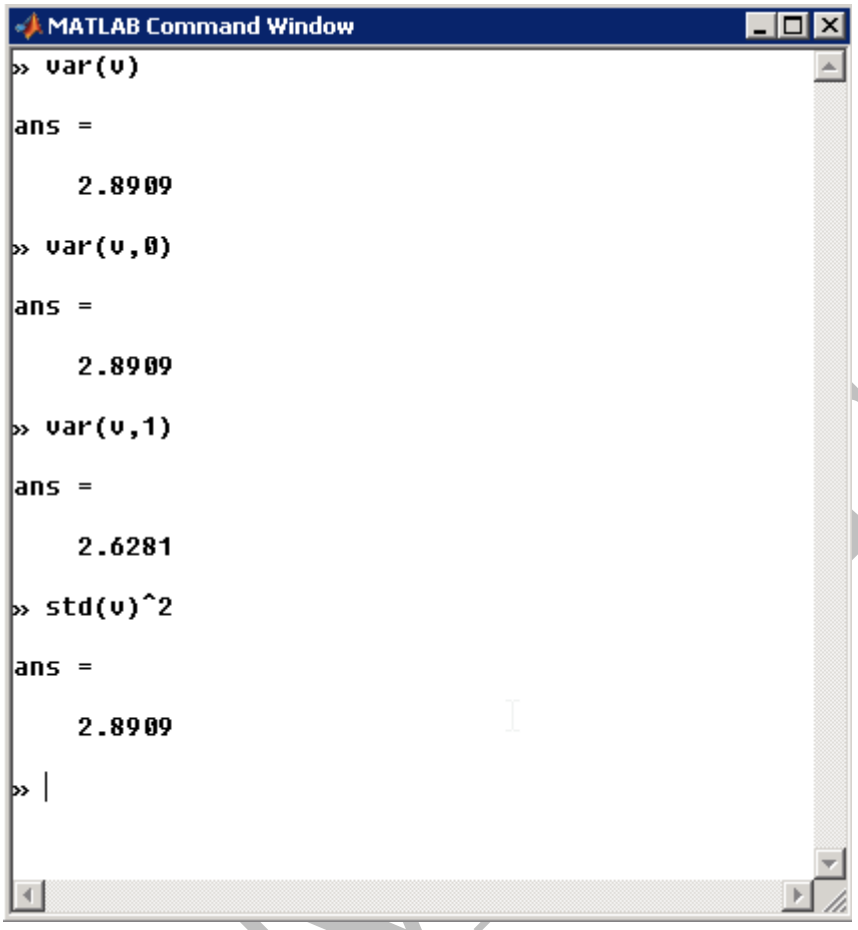
Now let us see if the arithmetical mean is available and correct with the **mean( )** function... and also the median with the **median( )** function and the mode with the function **mode( )**:



We must have a look the standard deviation and be sure that MATLAB use by default the non-biased estimator of the standard deviation with the function **std( )** and that we can choose the type of standard deviation we want (biased are non-biased):

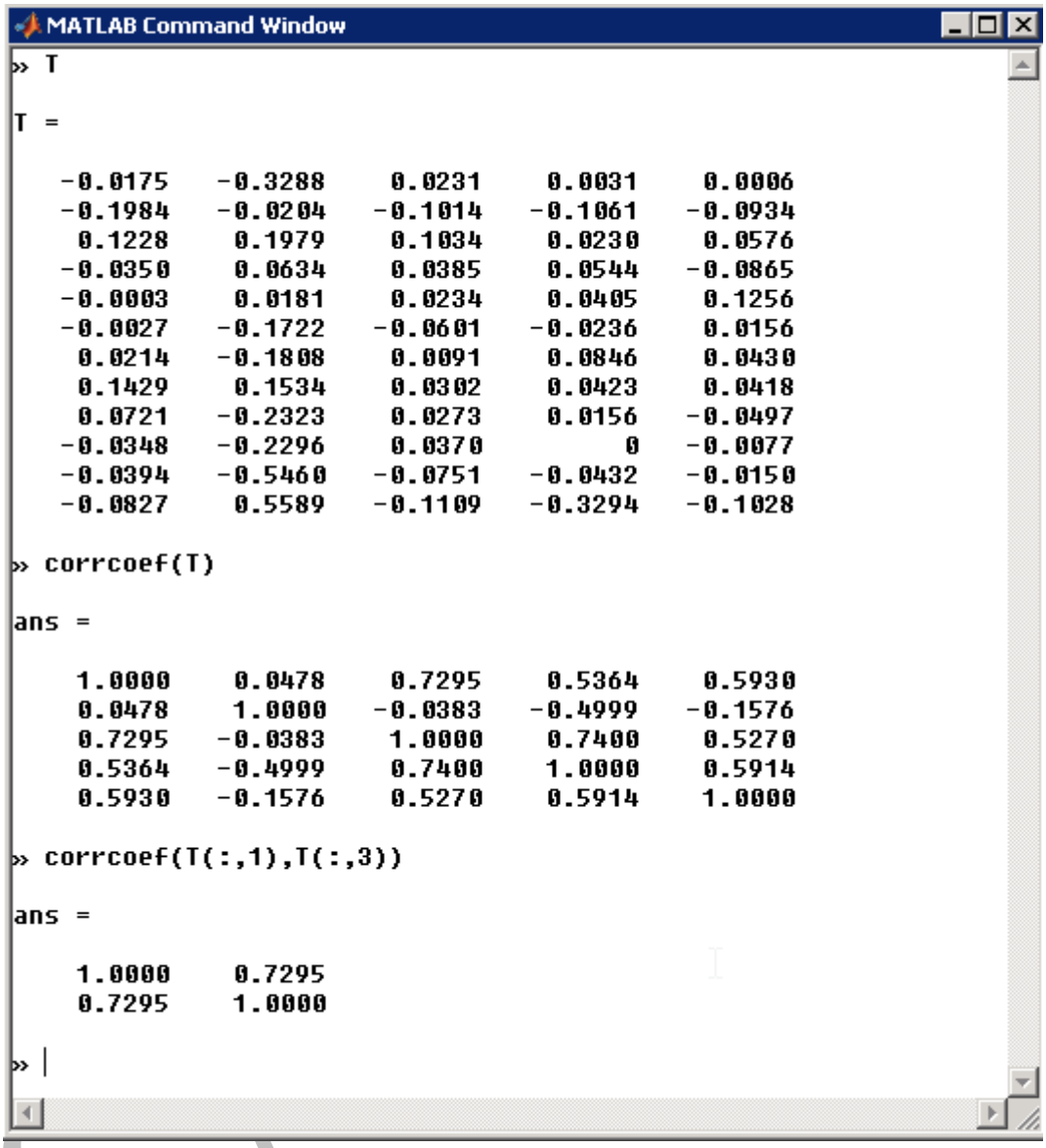


We can also check that the variance have the same flag with the function **var( )** to have the non-biased and the biased version:

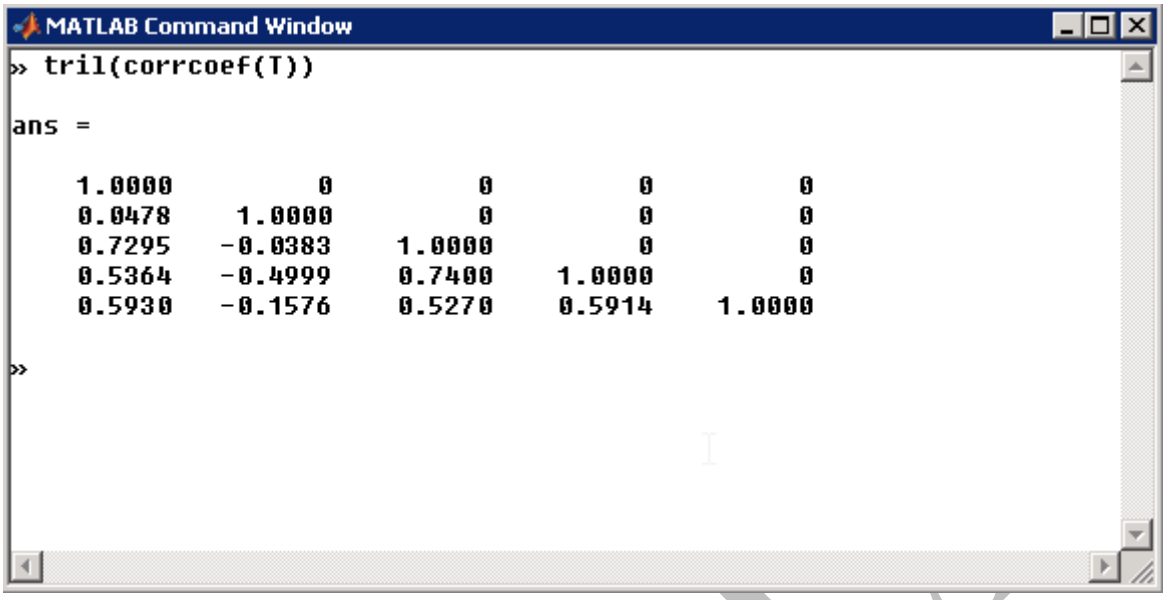


SAMPLE

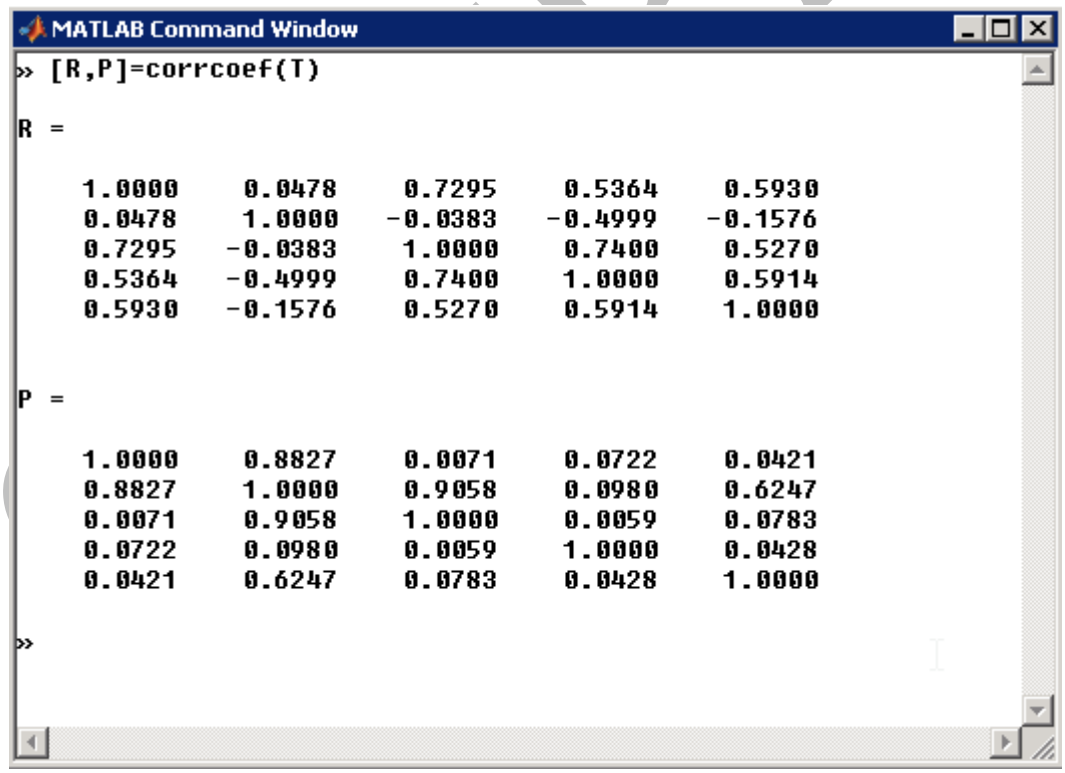
The correlation matrix is by default the non-biased correlation matrix when you use the function **corrcoef( )** on a whole matrix or on one of its subset:



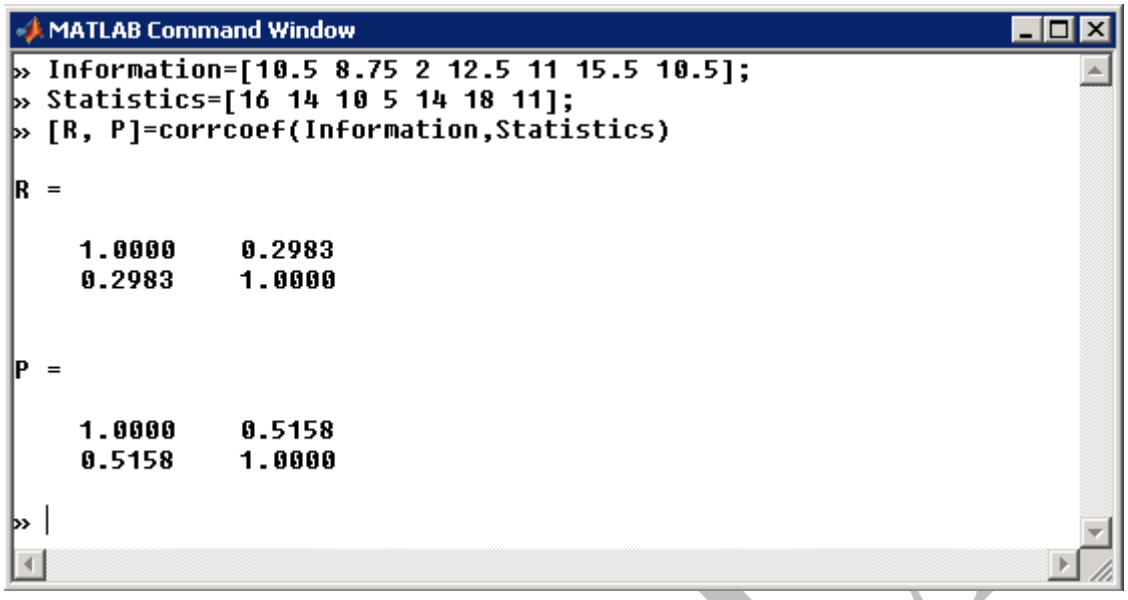
For sure we can apply the **tril( )** function together with the **corrcoef( )** function if we want:



Or by extracting the second output of the function **corrcoef( )** we can get the p-value of the Pearson test with by default a 5% treshold:



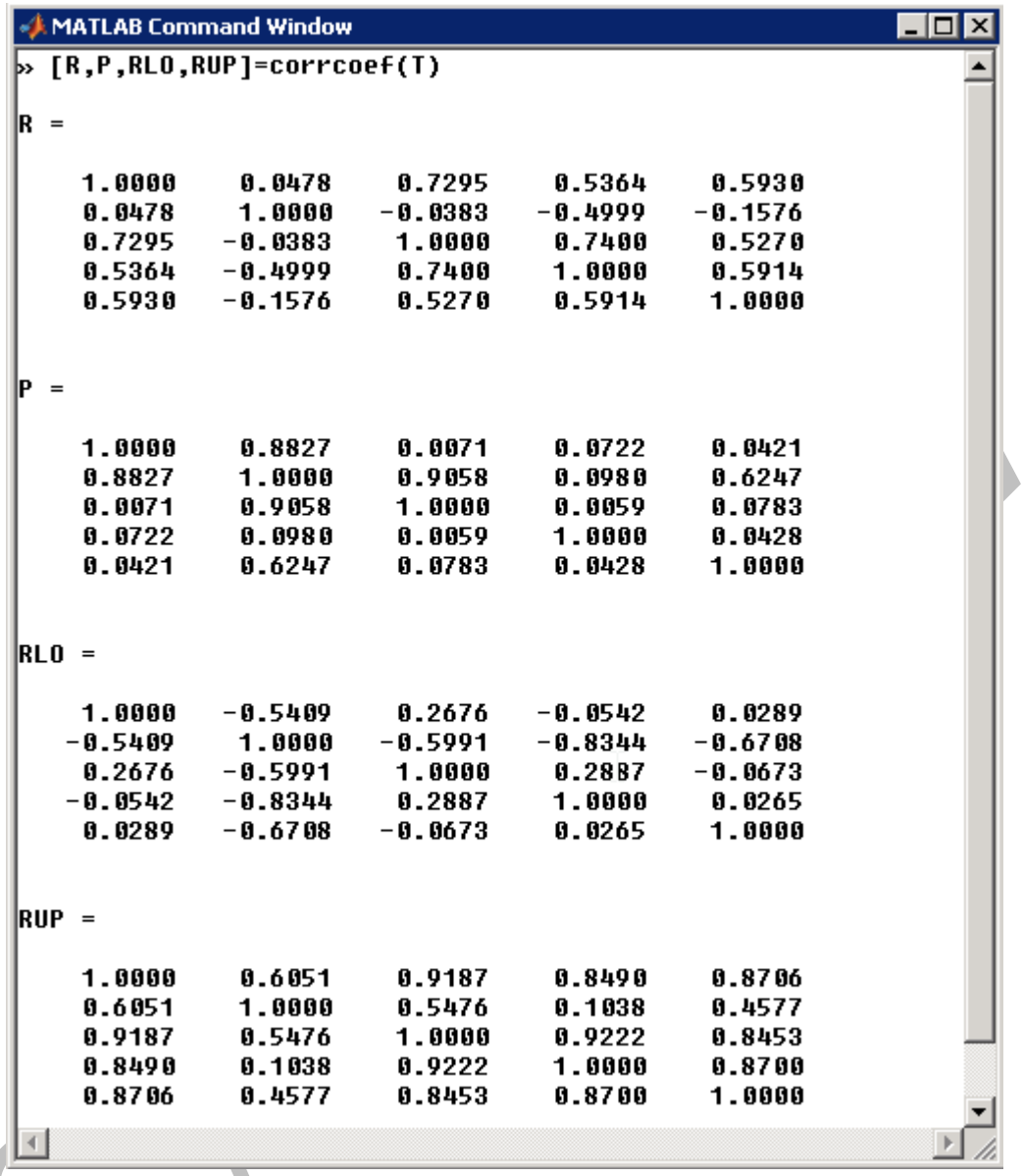
Let us check if the result corresponds to calculation made by hand in the Statistics course:



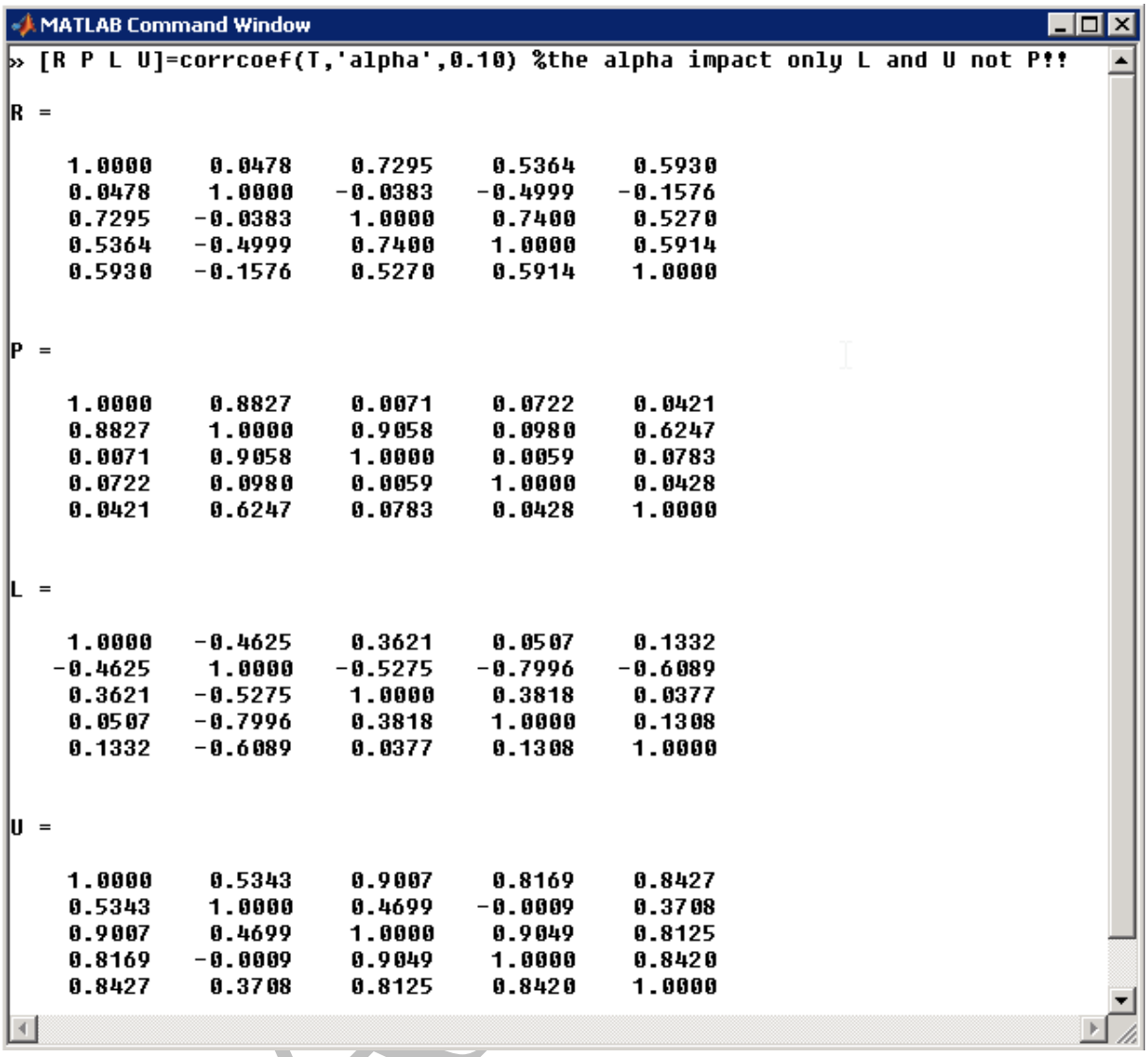
Yep! We get it!

SAMPLE

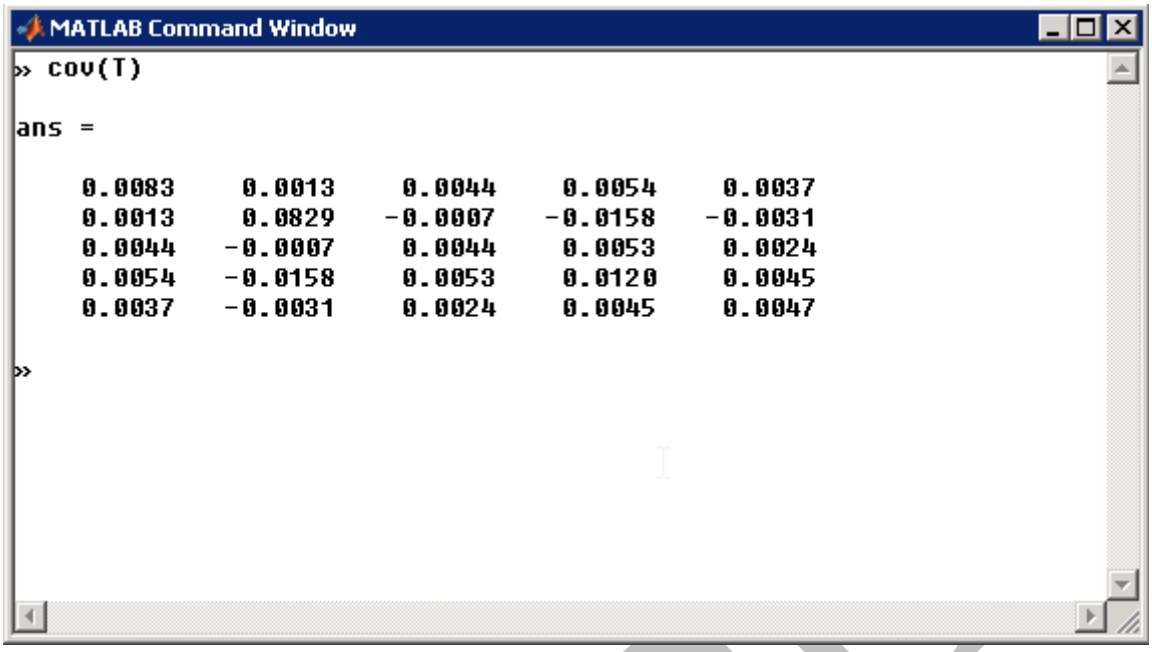
We can also have the lower and upper limits of the 95% confidence interval:



Or chose another value for the alpha threshold as the default one of 5%:



The non-biased variance-covariance matrix will be calculated thanks to the **cov( )** function:



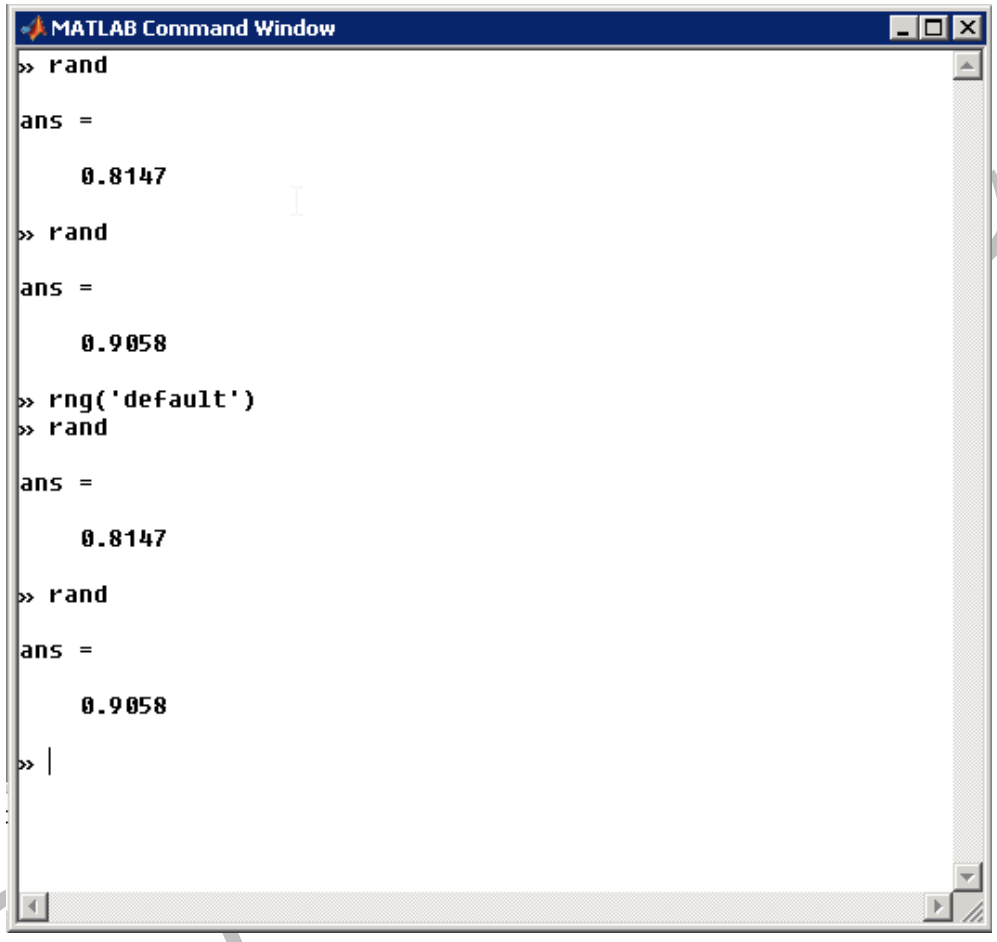
It's a shame that there is no option that let us choose if we want the biased or non-biased covariance matrix!

SAMPLE

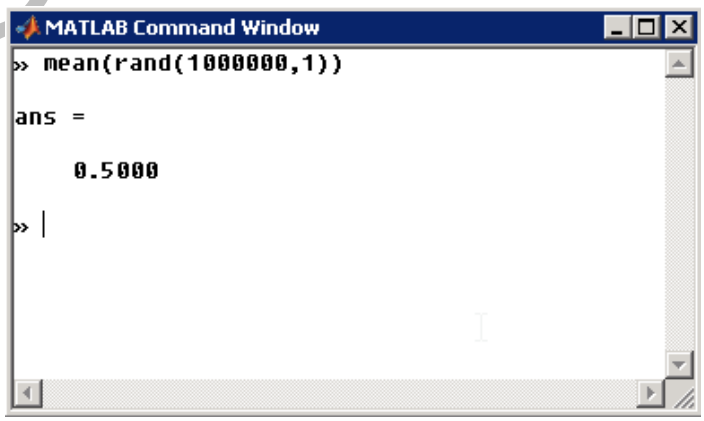
Random variable generation

Generate and manage random variables in MATLAB is a huge subject especially if you don't have the Statistical Toolbox... We will see now the idea behind what is available in the core version of MATLAB.

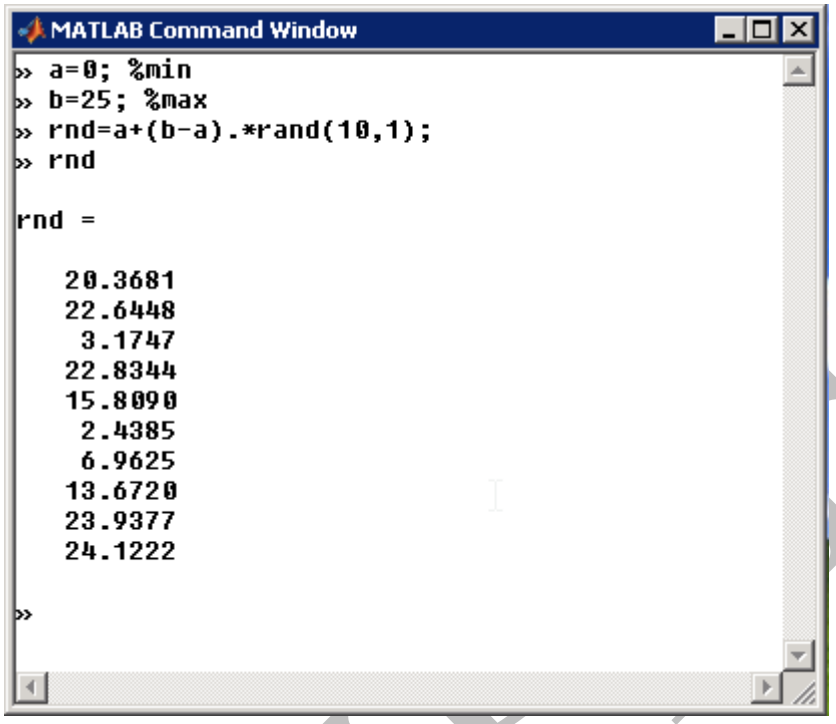
First the **rand** command by default will give you a uniform distributed random variable. If you want to restart the random generator use the command **rng('default')**:



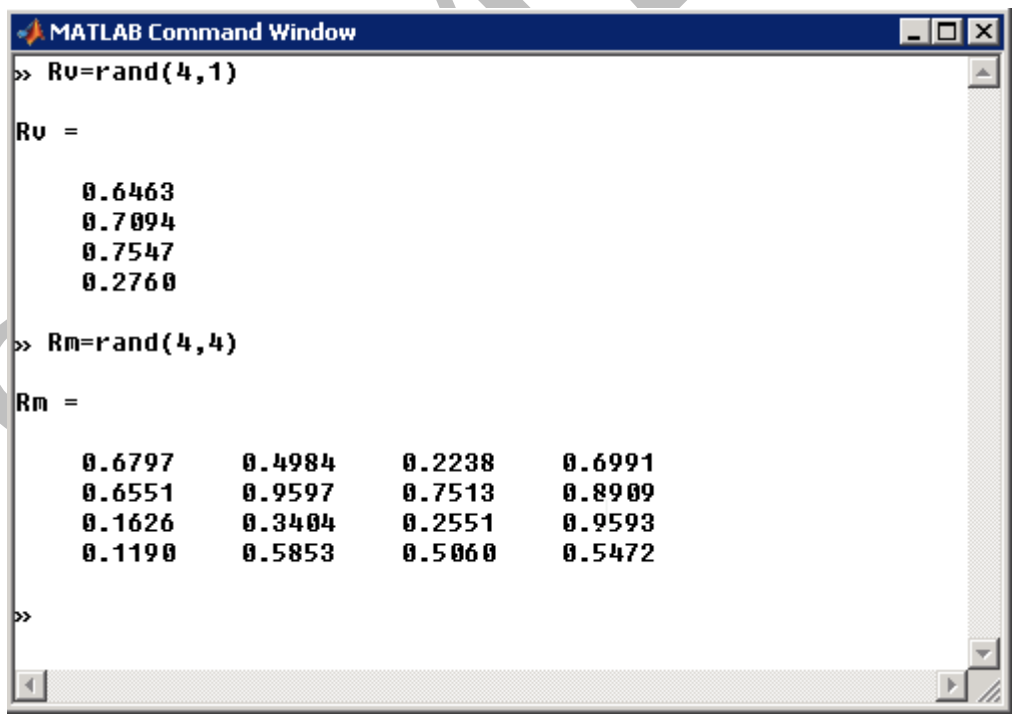
The rand is also a function **rand( )**. You can pass it the number of rows and columns of random variables that you want:



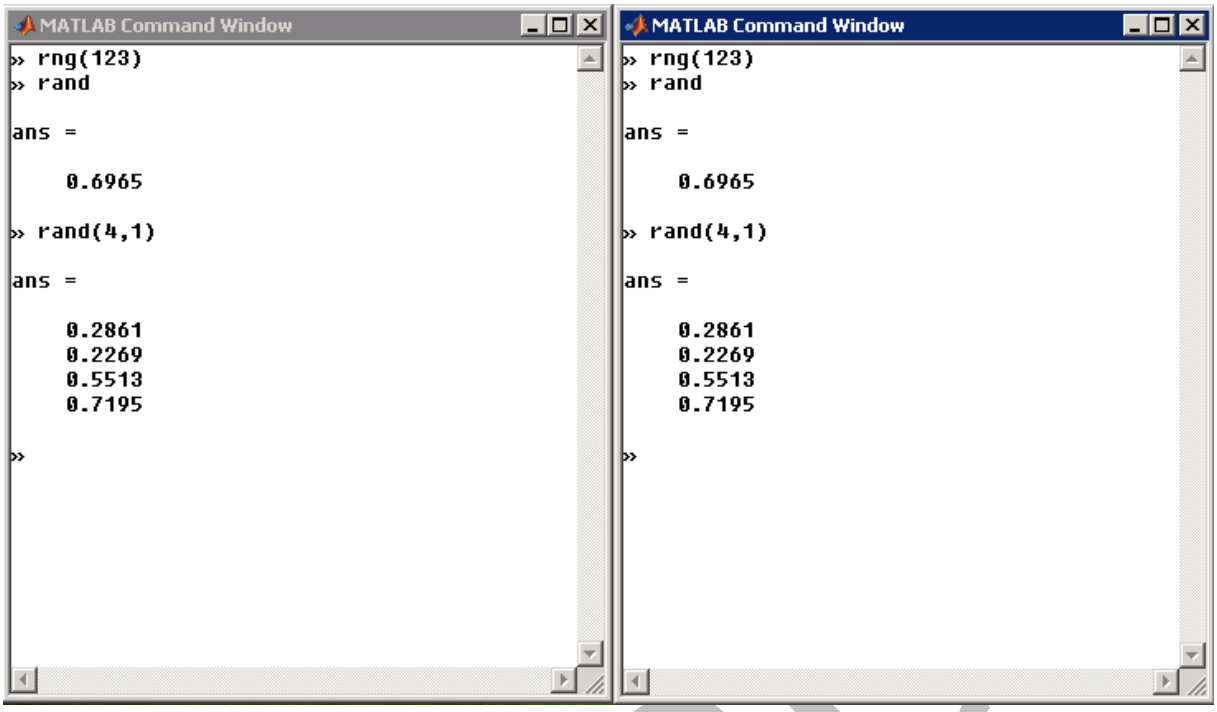
As far as I know there is no native function to simulate a uniform random variable in a given range. For this purpose you the following approach:



We can generate then vector columns but also matrices of random values:

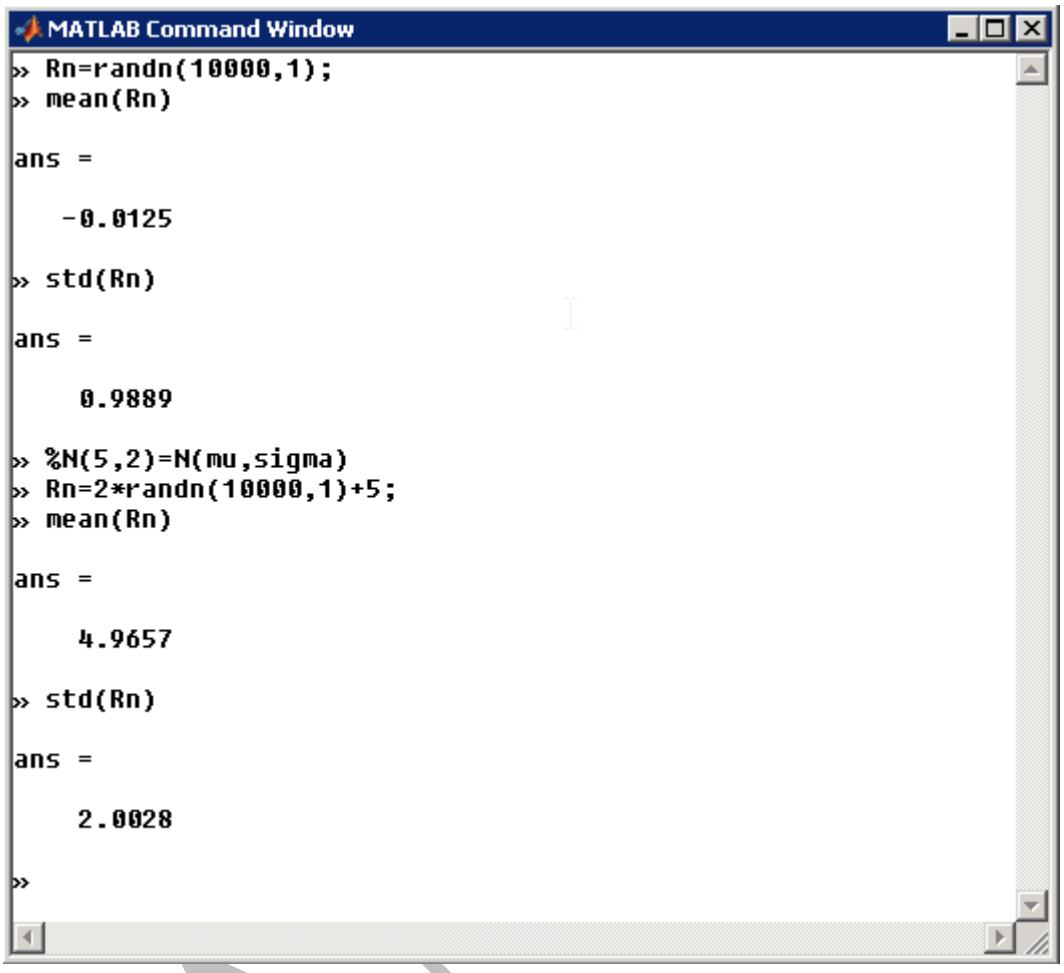


We can define a status for the random generator to be able to replicate the randomized values using the function **rng( )**:

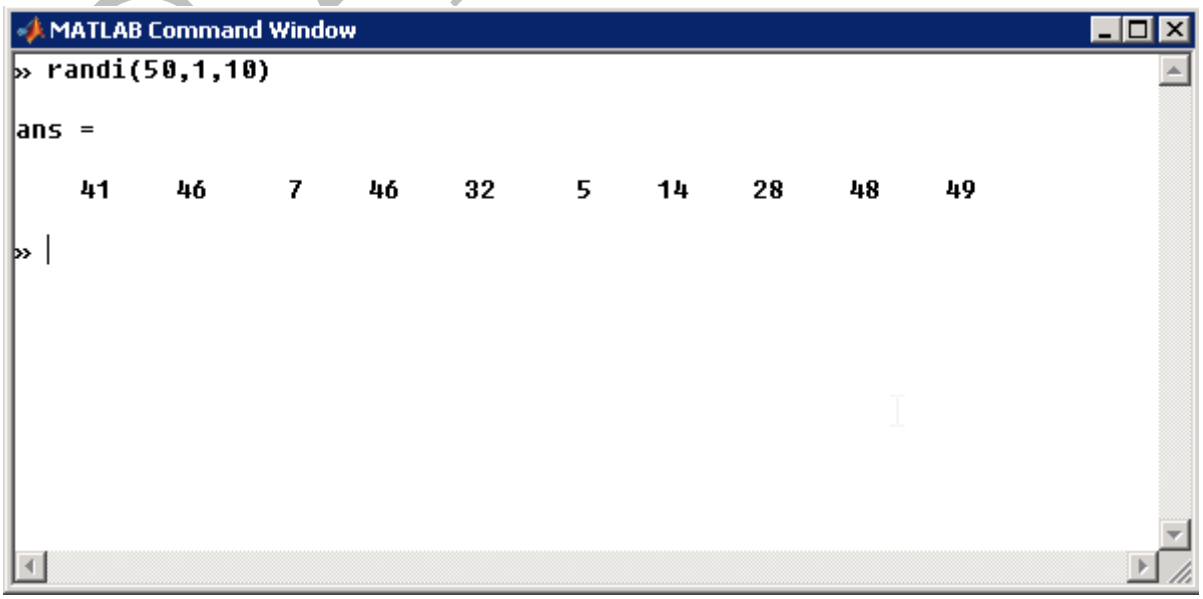

MATLAB Command Window
» rng(123)
» rand
ans =
0.6965
» rand(4,1)
ans =
0.2861
0.2269
0.5513
0.7195
»
MATLAB Command Window
» rng(123)
» rand
ans =
0.6965
» rand(4,1)
ans =
0.2861
0.2269
0.5513
0.7195
»

SAMPLE

The core version of MATLAB has also the possibility to generate and random Normal centered reduced variable with the function **randn( )** and if you use the you mathematical knowledge in the fact you can generate any Normal random variable:



You can also generate random integer value between two ranges (very useful!) with the function **randi( )**:



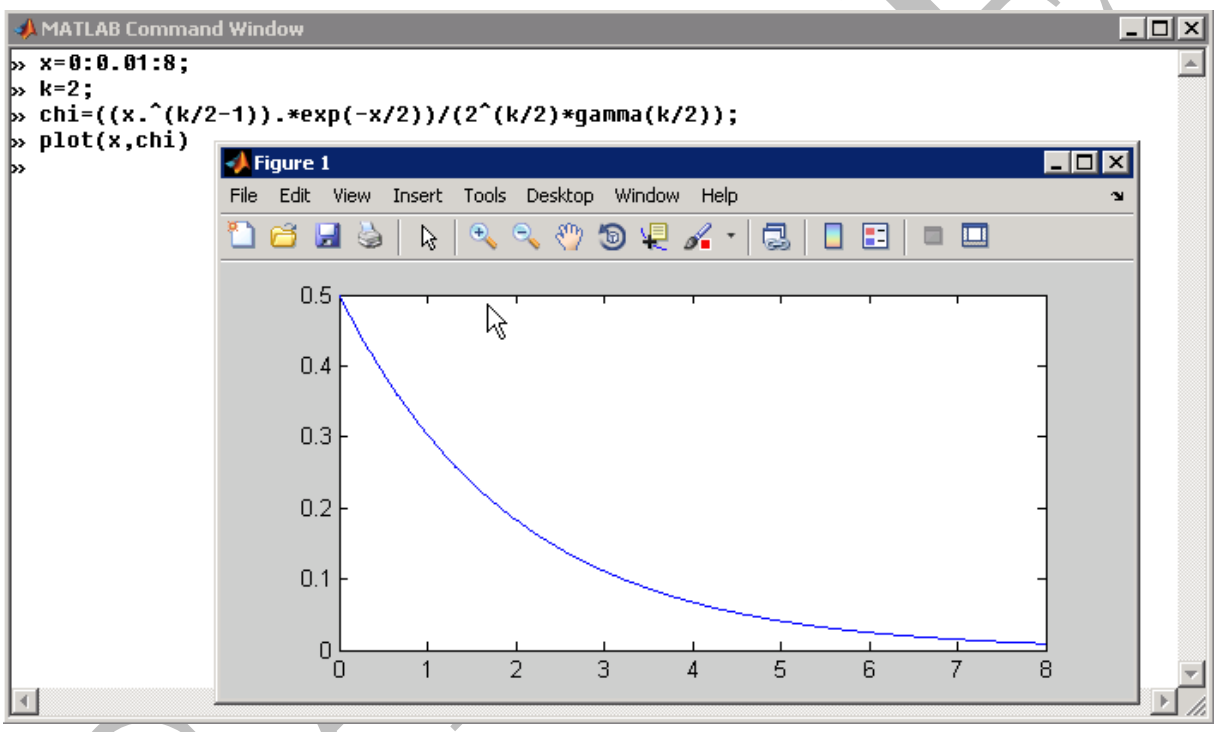
But what can you do if you want simulate a random variable that is out of the box of the MATLAB core version?

Then you just have to play with you knowledge of statistics. Imagine we want of example to simulate a chi-square random variable.

We know that:

$$\chi_k^2 = f(x,k)_{[0+\infty[} = \frac{x^{k/2-1}e^{-x/2}}{2^{k/2}\Gamma\left(\frac{k}{2}\right)}$$

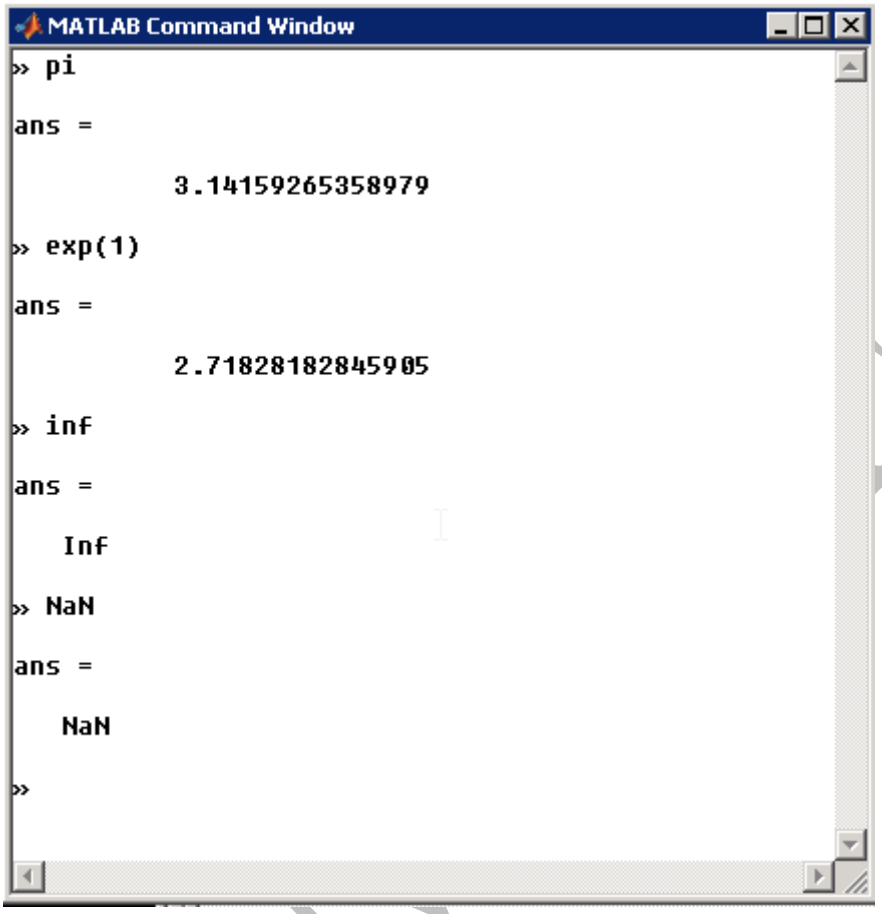
Thus:



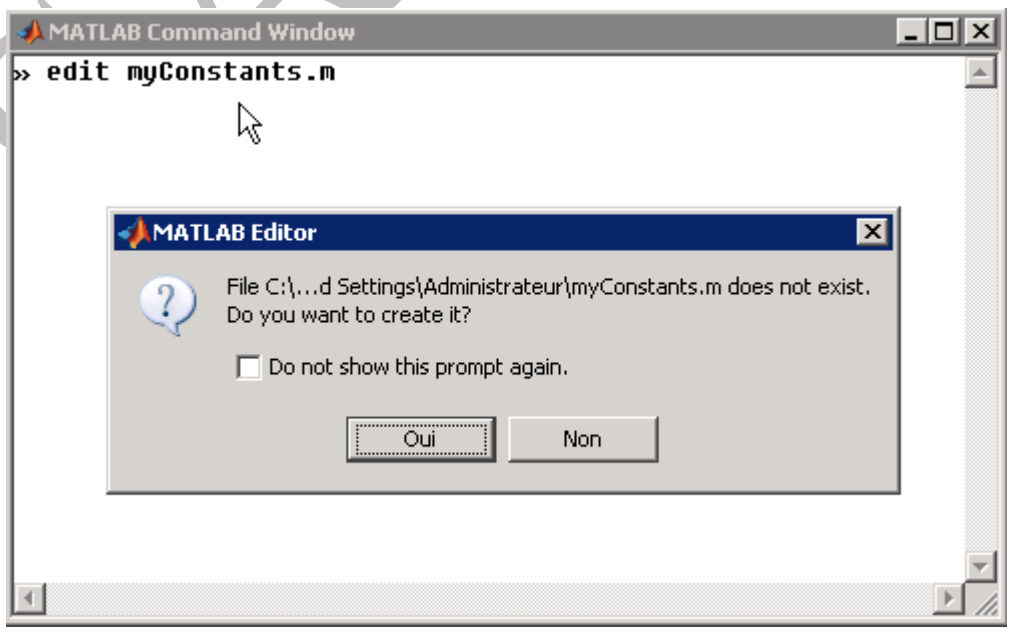
and so on for any other rand variable...

Use of constants (permanent variables)

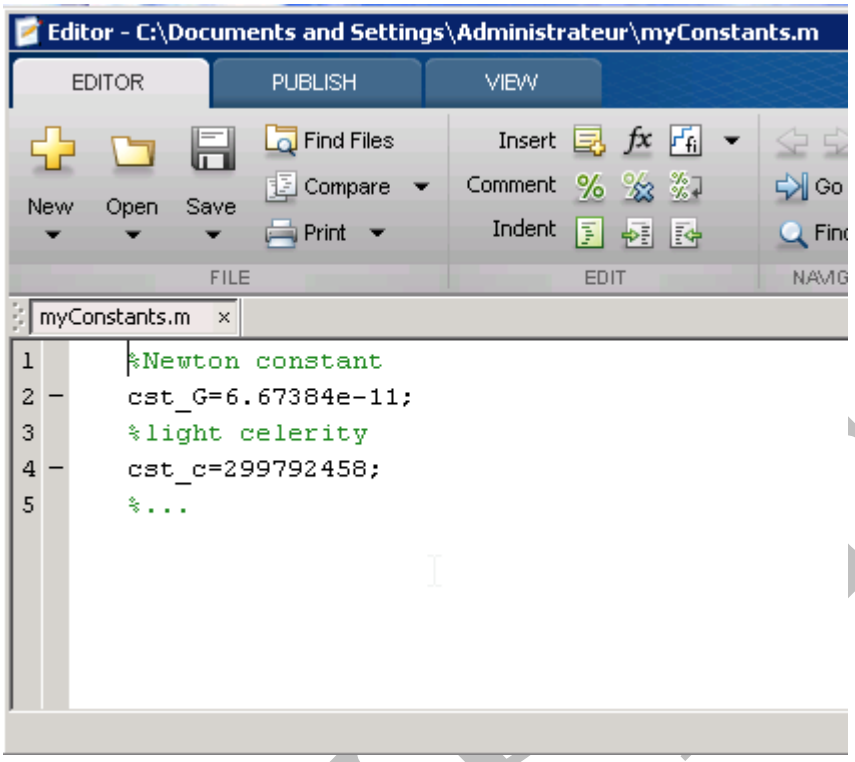
In MATLAB there are a few mathematical constants only:



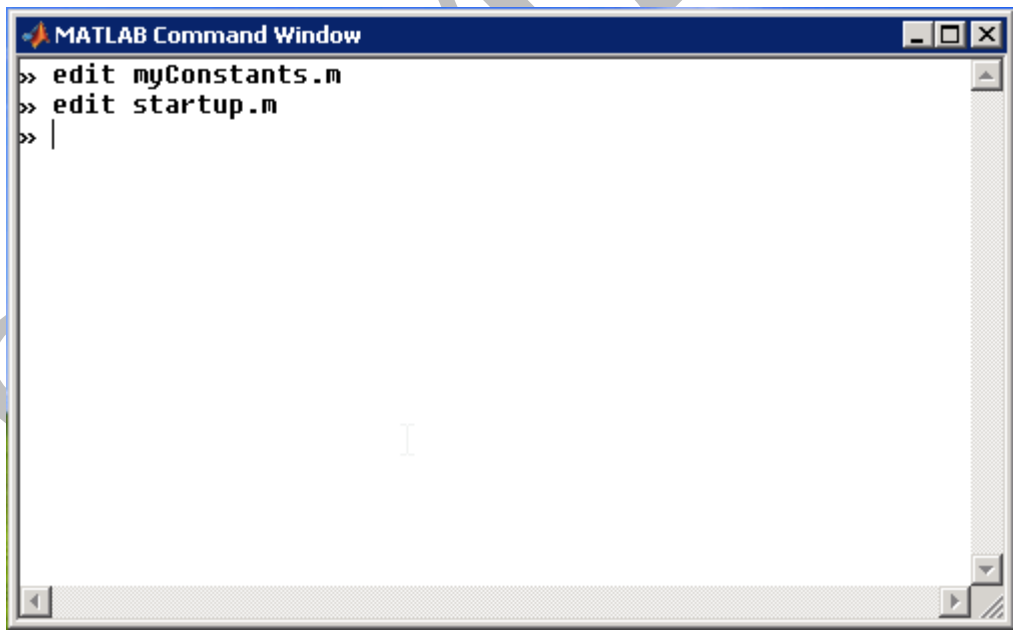
If you are a physicist this is an issue. A workaround is to create a new script file (the name doesn't really matter):



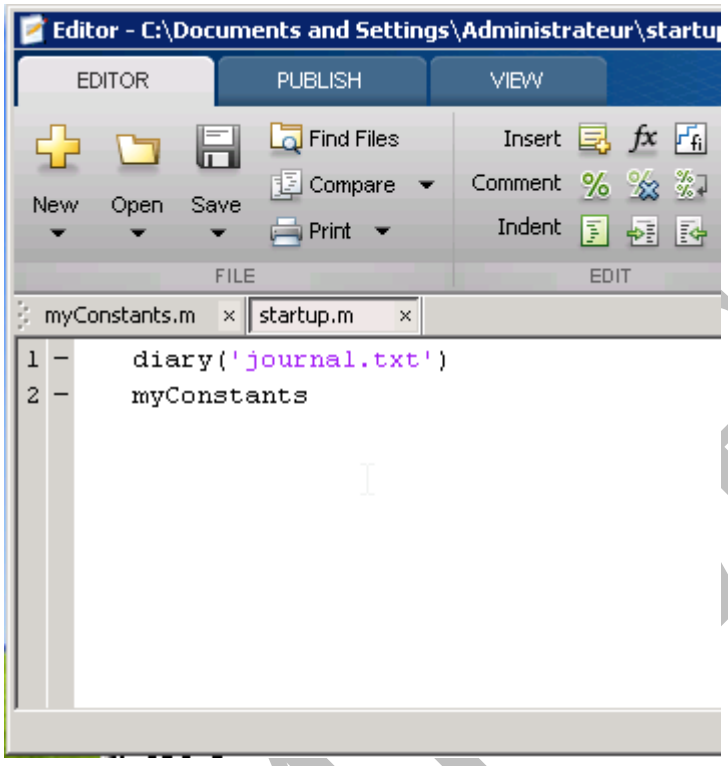
Type in all the constants you use for your work:



Once saved, open the file *startup.m*.



And call your script with your constants just by writing the script file name (if you don't specify the path that means that your file is in a MATLAB path):



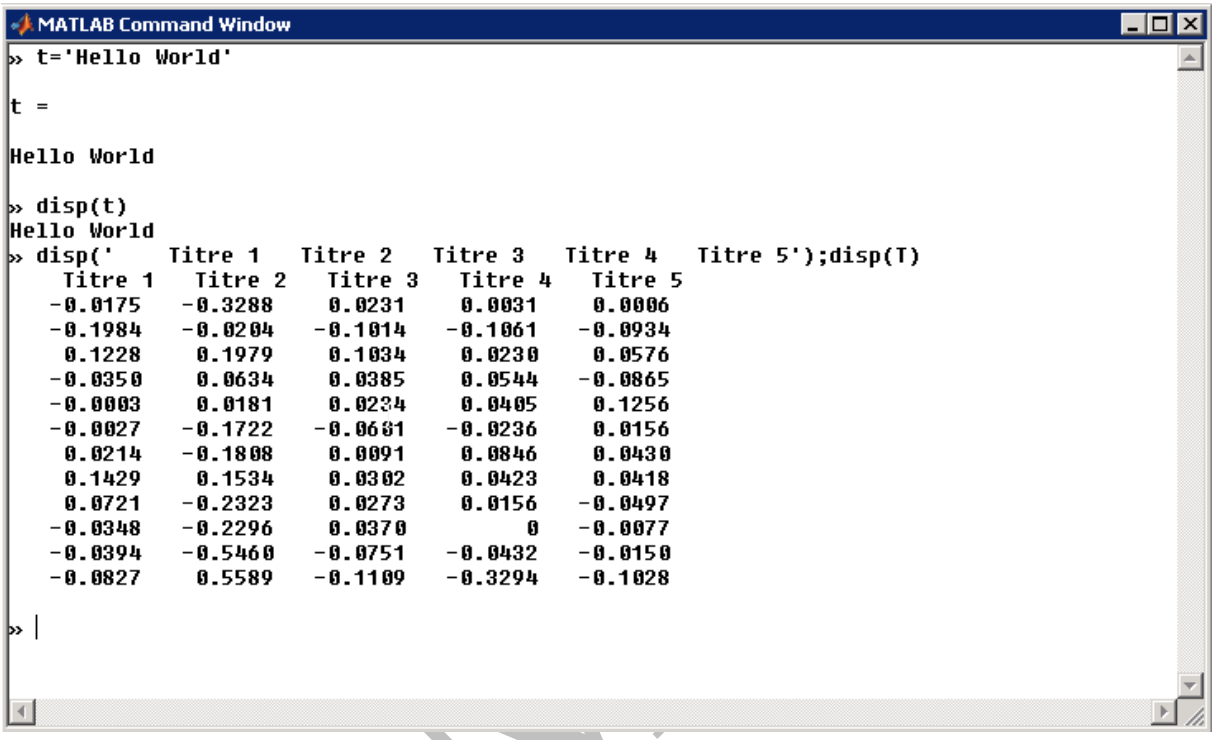
And the job is done!

SAMPLE

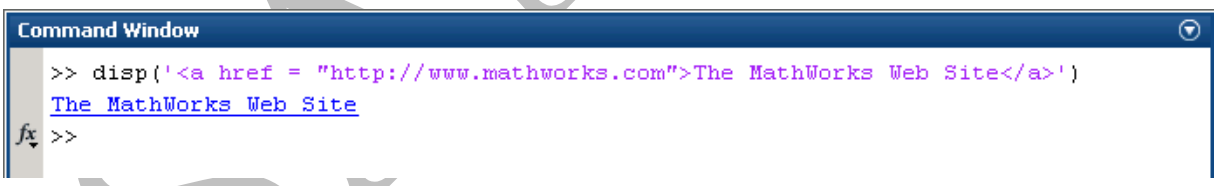
Working with strings

Working with strings is very important especially when you work with charts or that you automate reporting.

One of the first point to learn is to know different ways to add title rows to a matrix using a small trick with the function **disp( )**:

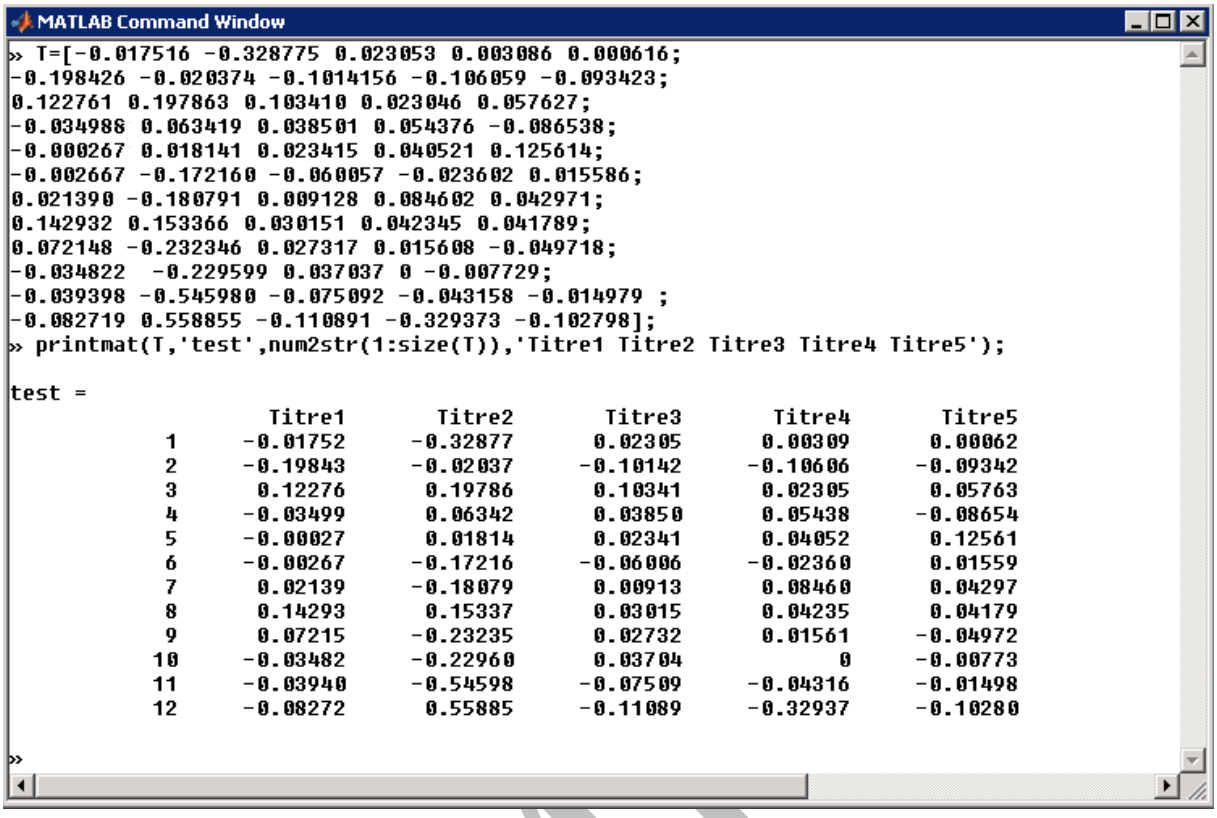


Or if you want to display a link (very useful for documentation):

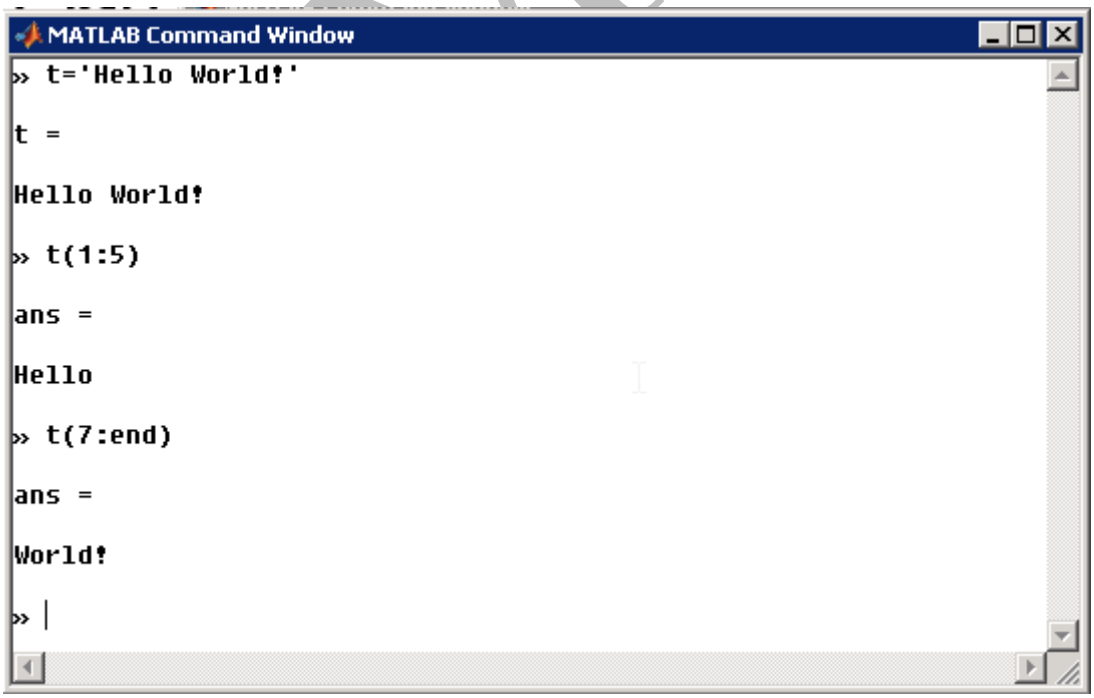


Note: This doesn't work in the old style Command Window!!!

There is another official way to do the same using the function **printmat( )**:

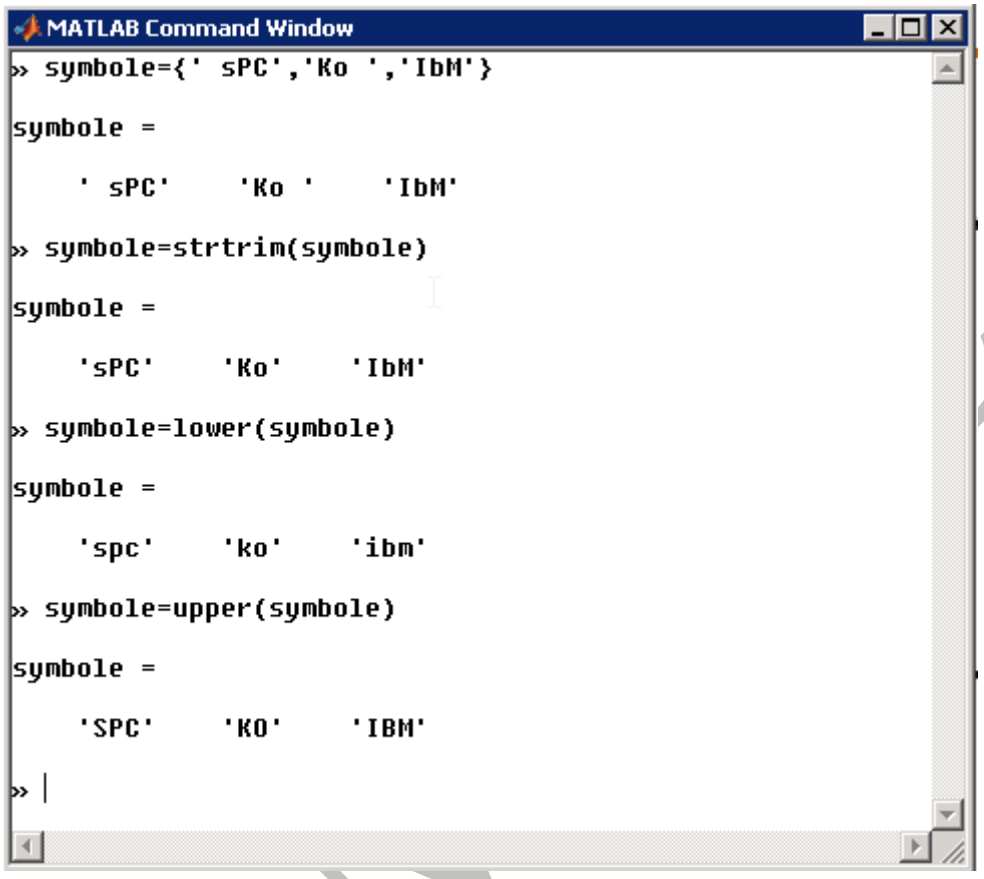


If we want to extract a substring of string remember that strings are also vectors by default:

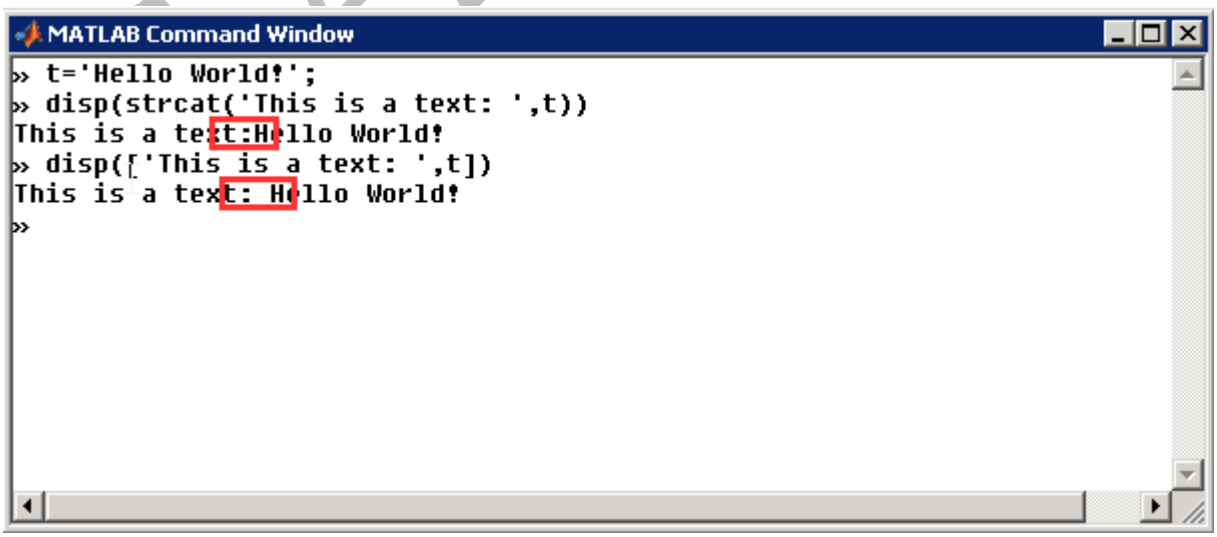


Now what about manipulating cell array that by default are made to contain strings?

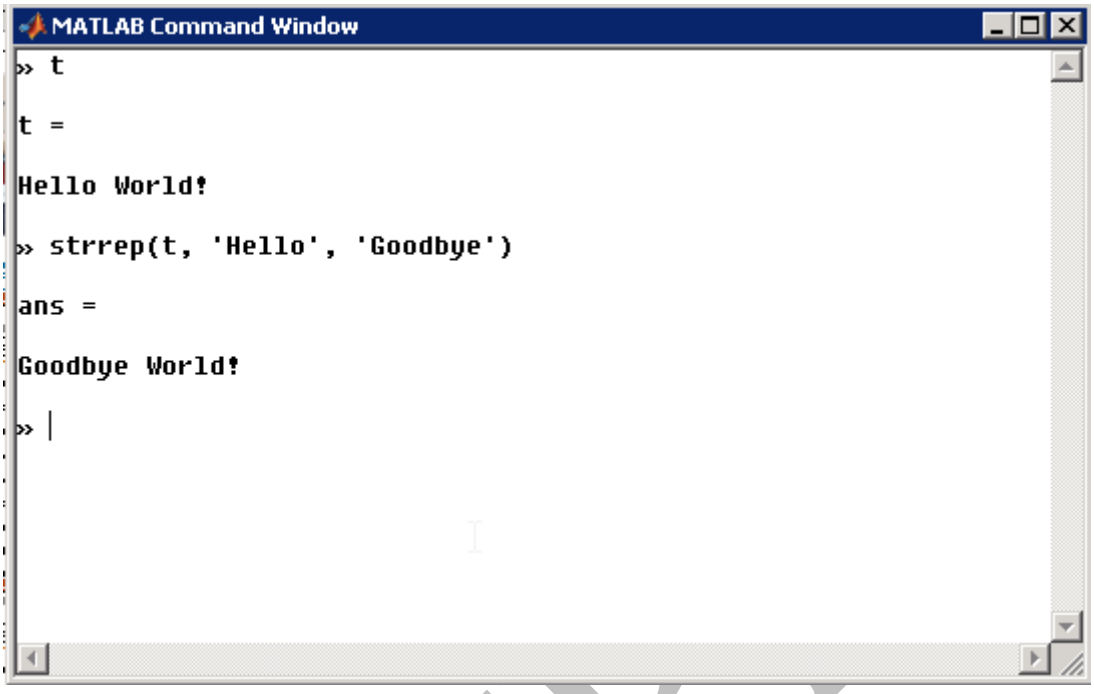
First we can remove non desired spaces using the **strtrim( )** function or force lower or uppercase respectively with the functions **lower( )** or **upper( )**:



A typical operation is to concatenate strings together. If you do that using the both functions **disp( )** dans **strcat( )** the blank spaces will be removed as you can see below. But if you use a vector of strings then everything will be allright!:

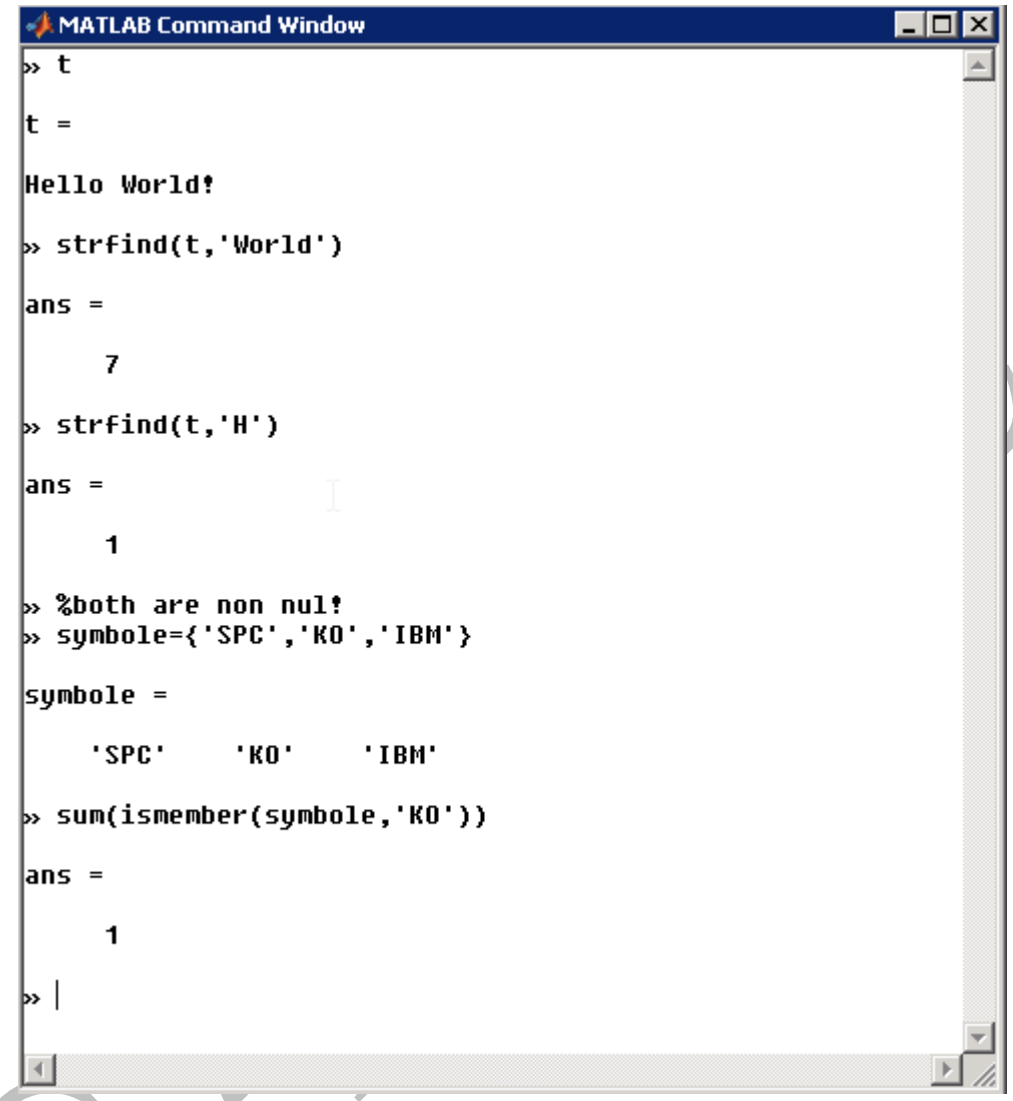


Something also useful sometimes is to replace a word by another one using the function **strrep( )**:

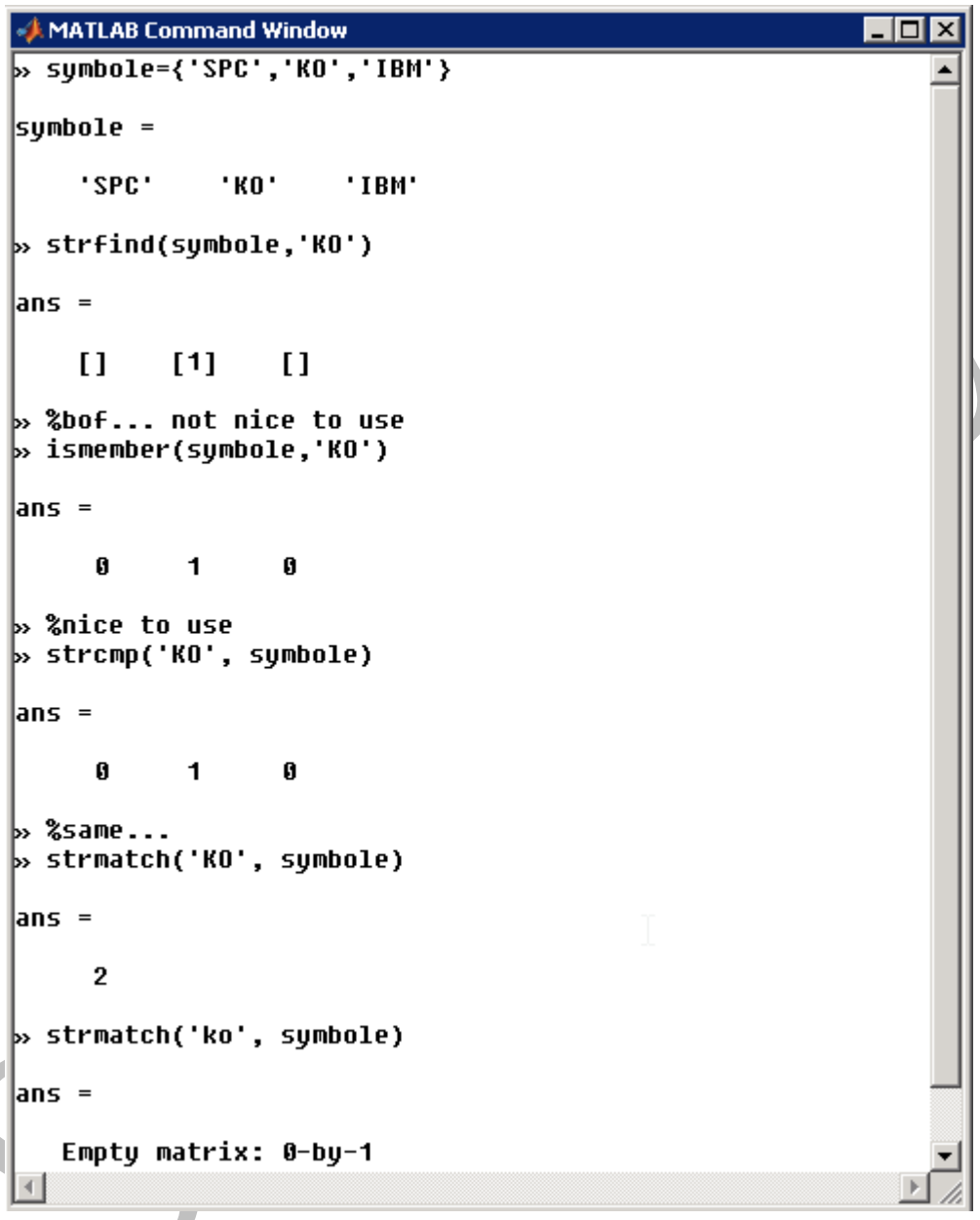


SAMPLE

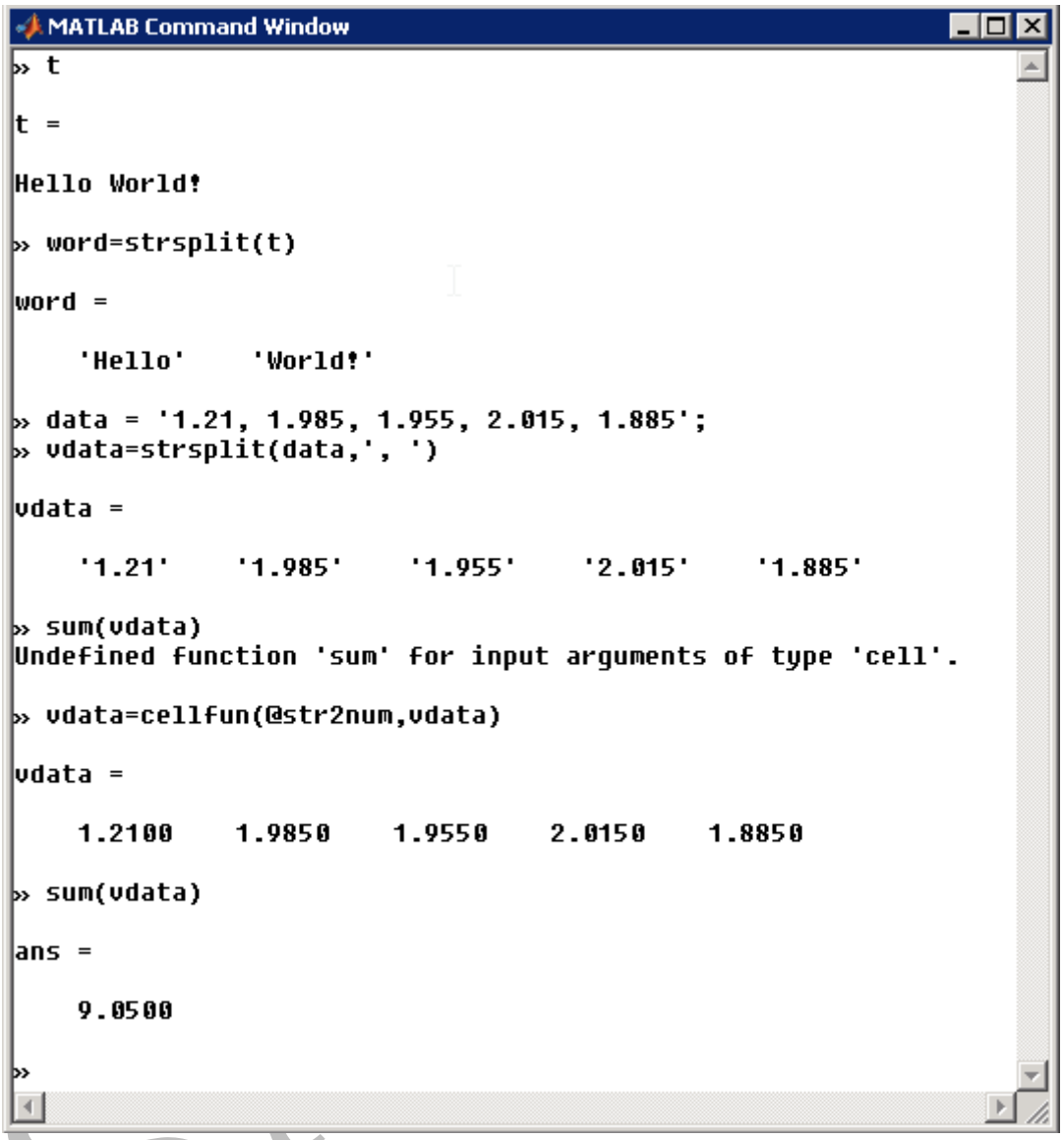
Sometimes it can be useful to search a character or a word in a string or even in if it exists in a cell array. You can achieve this respectively thanks to the functions **strfind( )** and **ismember( )**:



As you can see we can also use **strfind( )** to search in a cell array but output is not very nice. Otherwise the functions **strcmp( )** and **strmatch( )** do almost the same with the difference that **strmatch( )** is case sensitive:



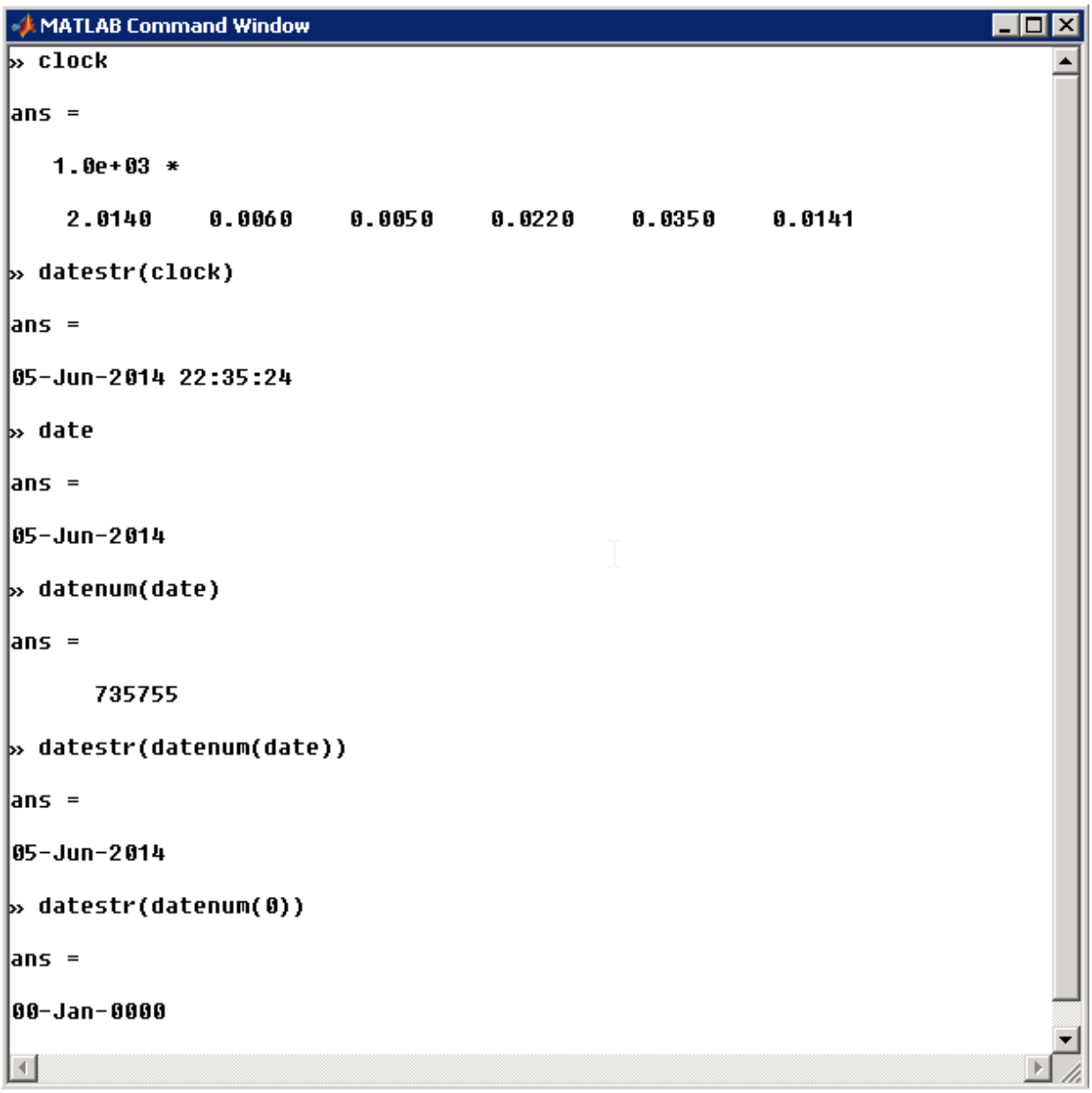
The function **strsplit( )** is also sometimes useful when you create GUI and a field where people can input multiples values at once. The problem is that you must not forget to convert the values to numerics using after the cell array function **str2num( )**:



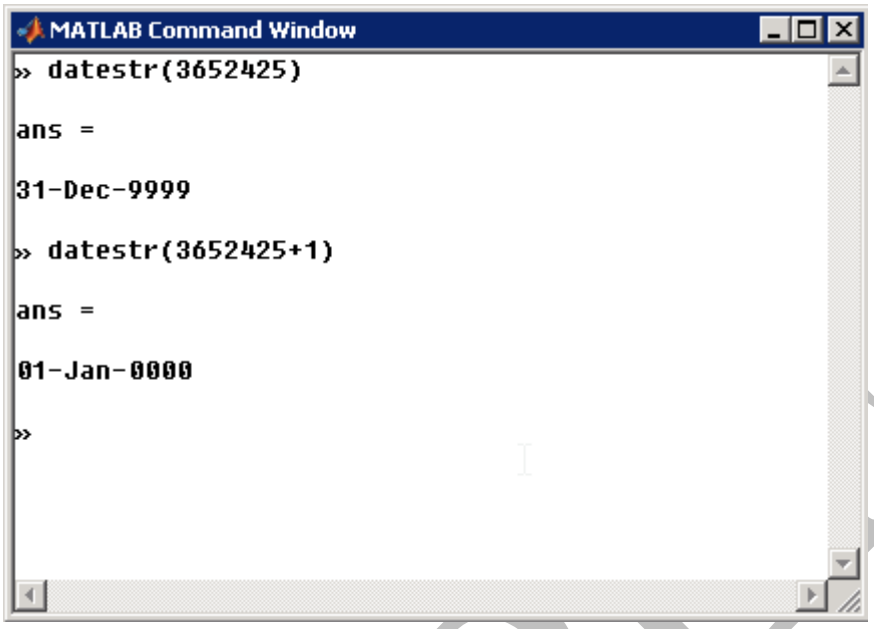
Working with dates

Working with dates is something also very important because most softwares and peoples around the world don't respect ISO norms...

Let us see first the different way to obtain the actual time using the commands **clock** and **date** together with the functions **datestr( )** and **datenum( )**:

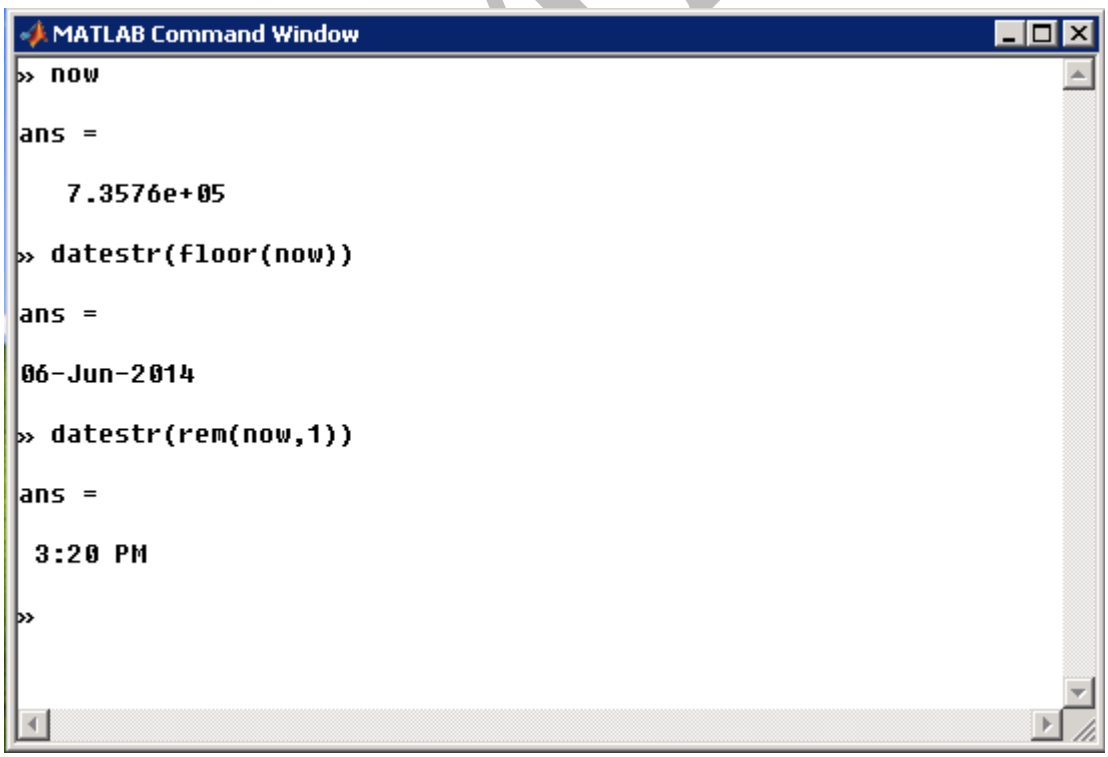


We can also have a look at the biggest and smallest date available in MATLAB by playing with **datestr( )**:

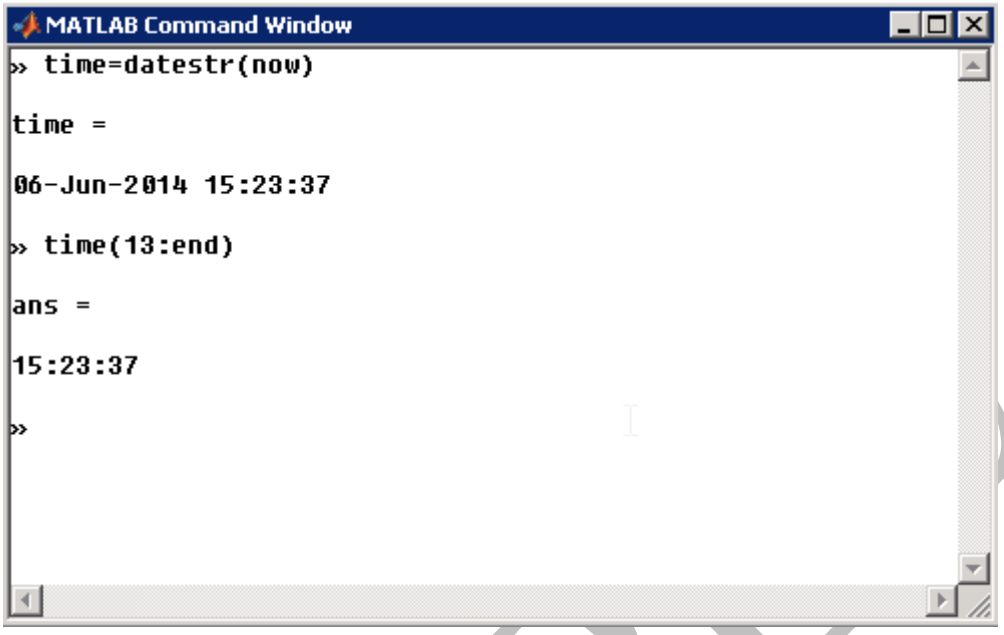


These poor limits ca be problematic for astronomers...!

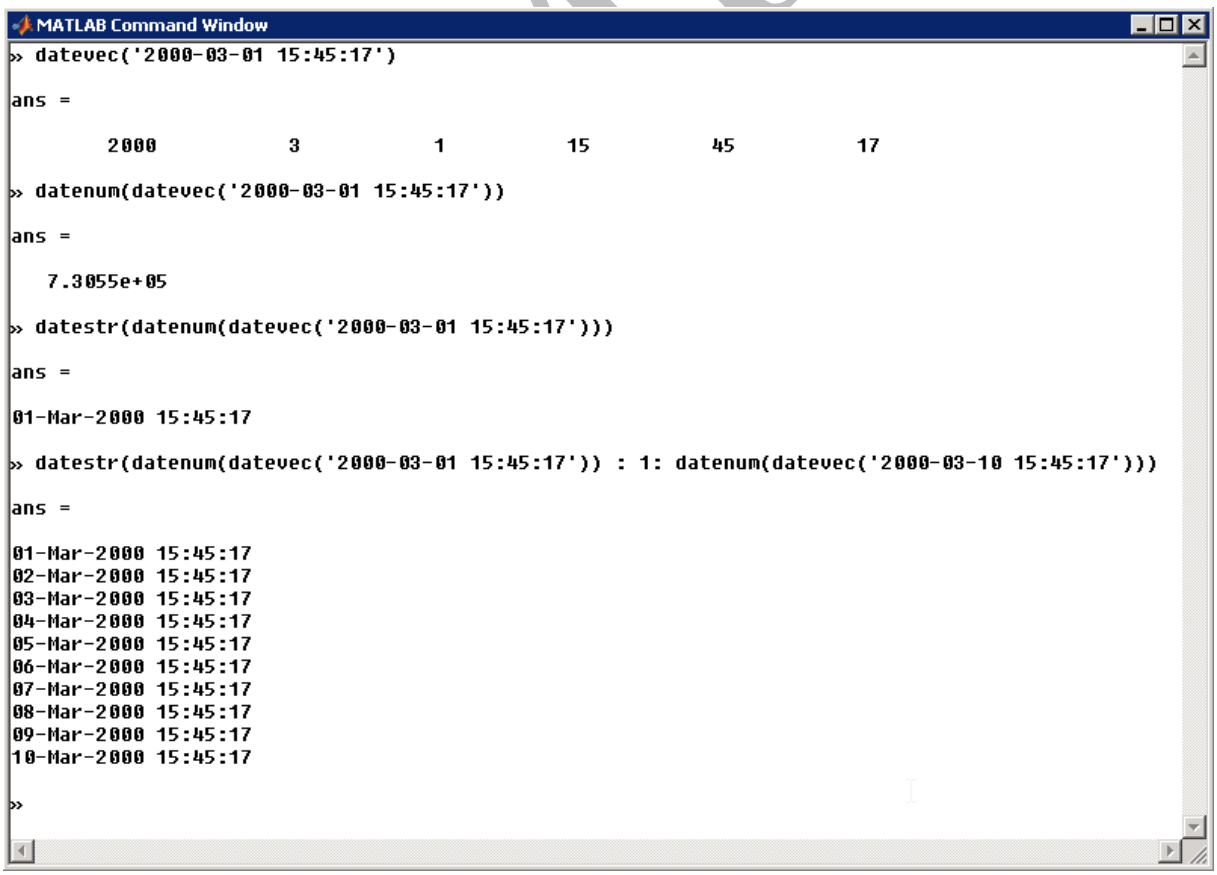
We can also play with the command now to get the actual date and time and if you want to get only the date or only the time you will have to play with the functions **floor( )** and **rem( )**:



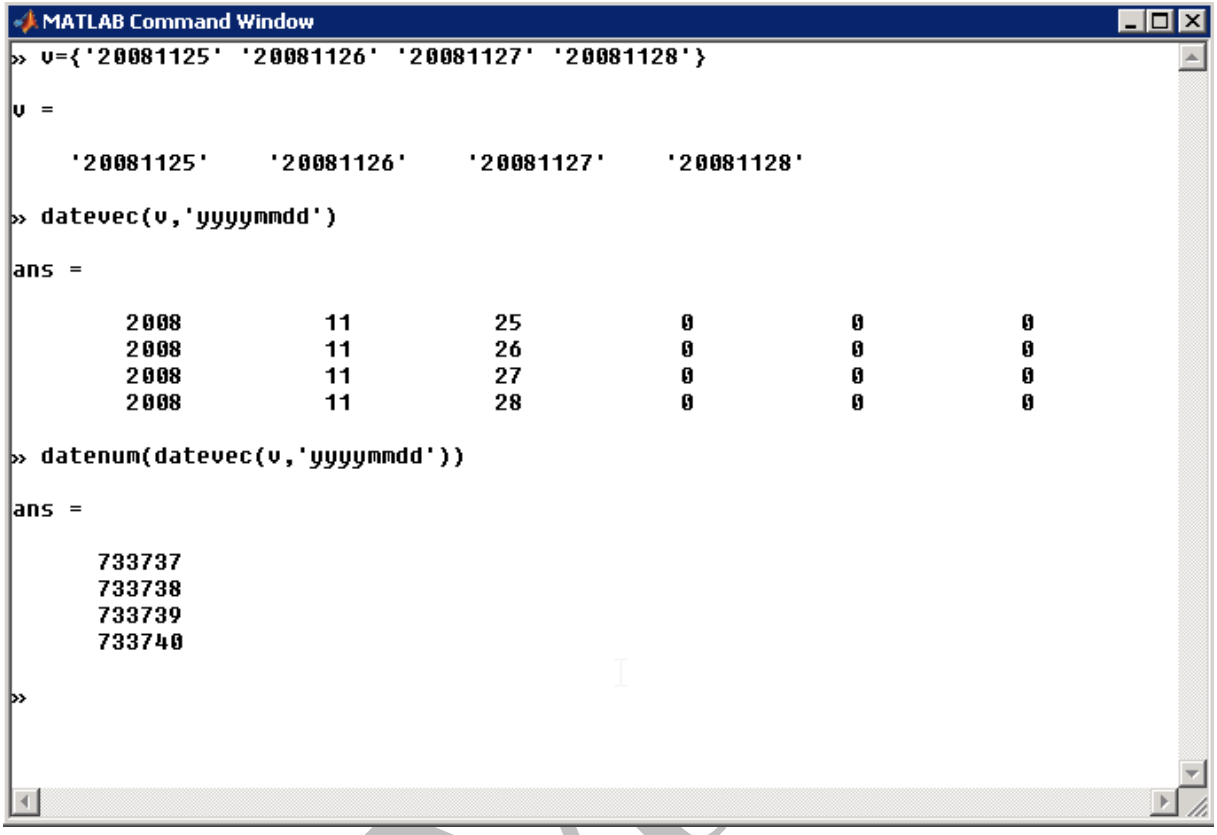
Here you can see the real full output of the command **now** and another way to get the time + the seconds:



It is also very interesting to see what happens when you pass a date trough the function **datevec( )** and also how to build a sequence of dates using **datevec( )** in association with **datenum( )** and **datestr( )**:

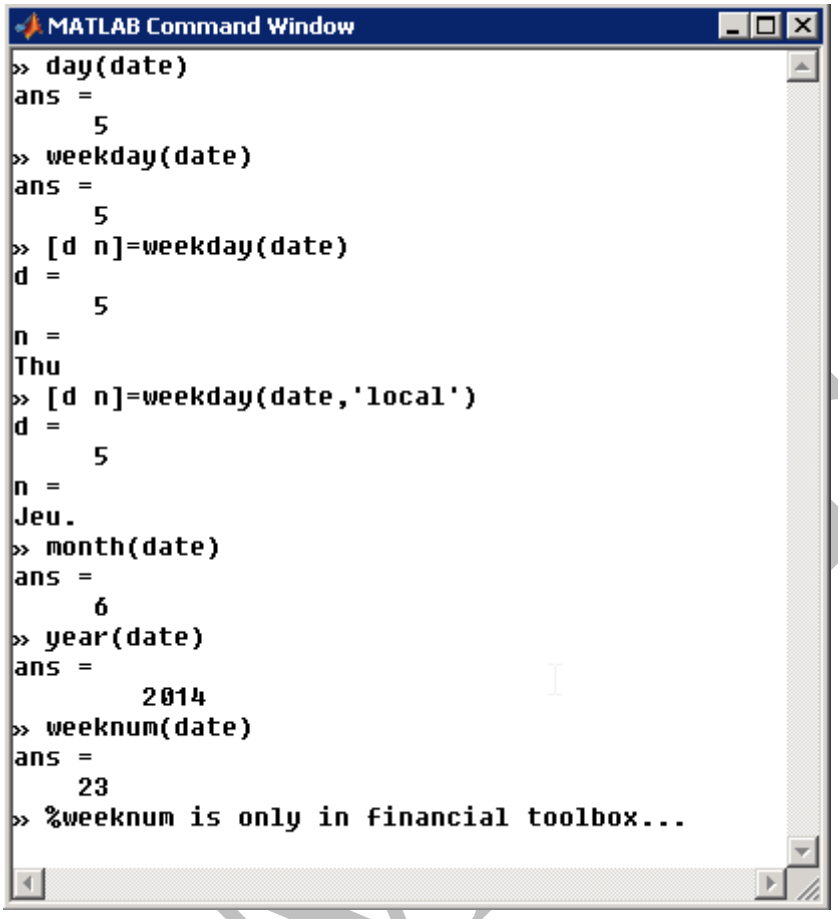


Even when the dates are given in some horrible format you can still give a formatting argument to the function **datevec( )** so that it can recognize the date pattern of you original datas:

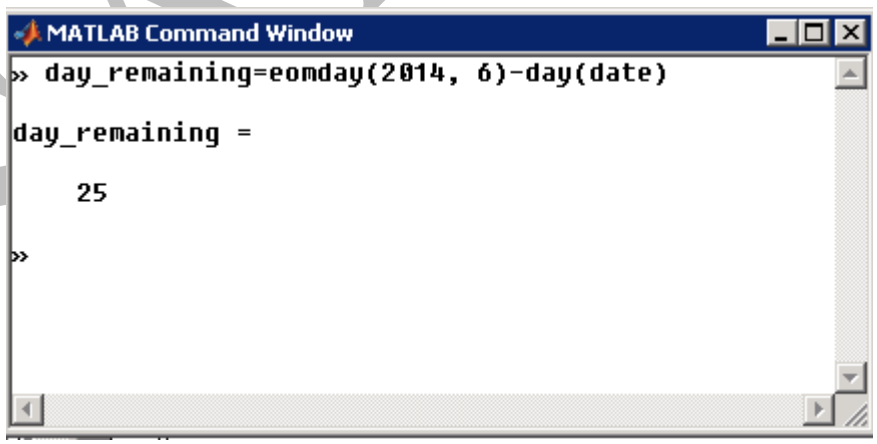


SAM

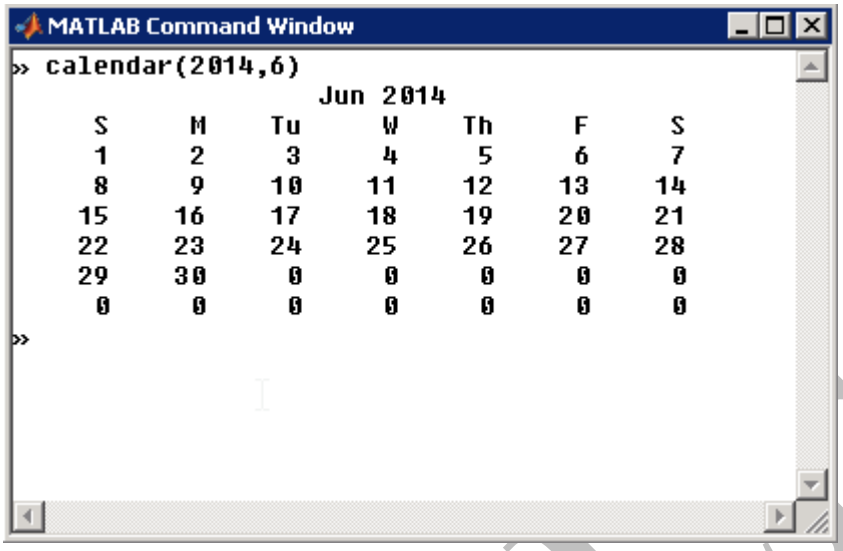
Like in Microsoft Excel or any other spreadsheet software the core version of MATLAB has sum elementary extraction dates functions like **day( )**, **weekday( )**, **month( )** and **year( )**:



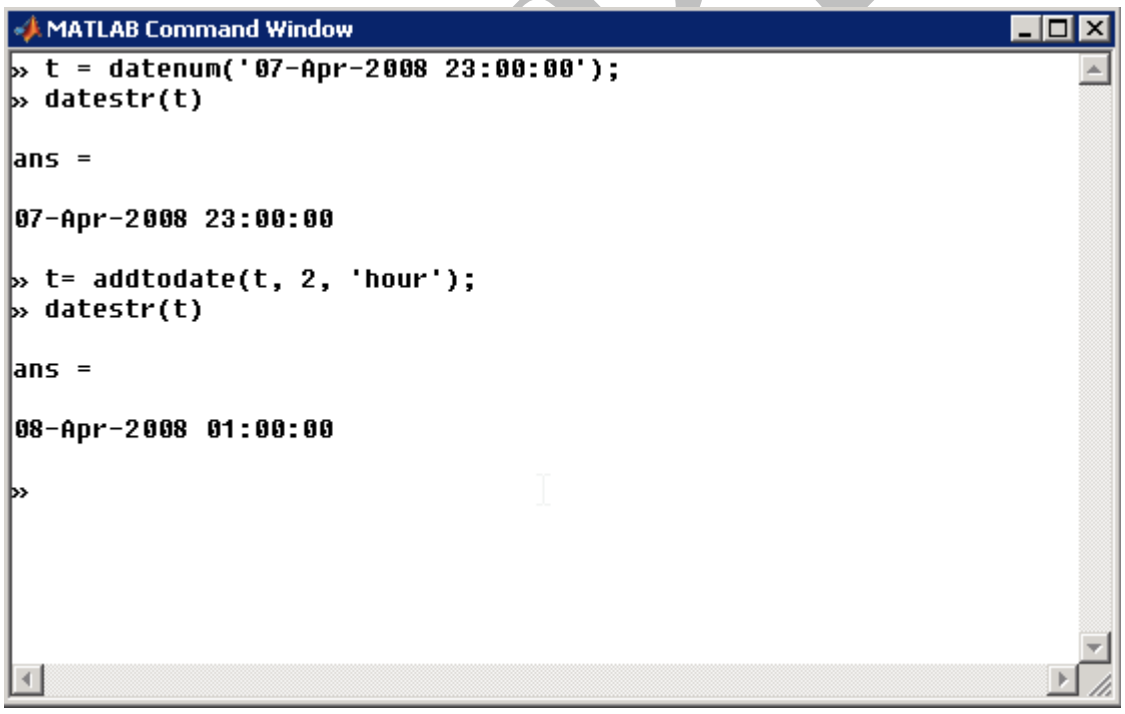
Another typical calculation is to calculate the number of days remaining until the end of the month using the function **eomday( )**:



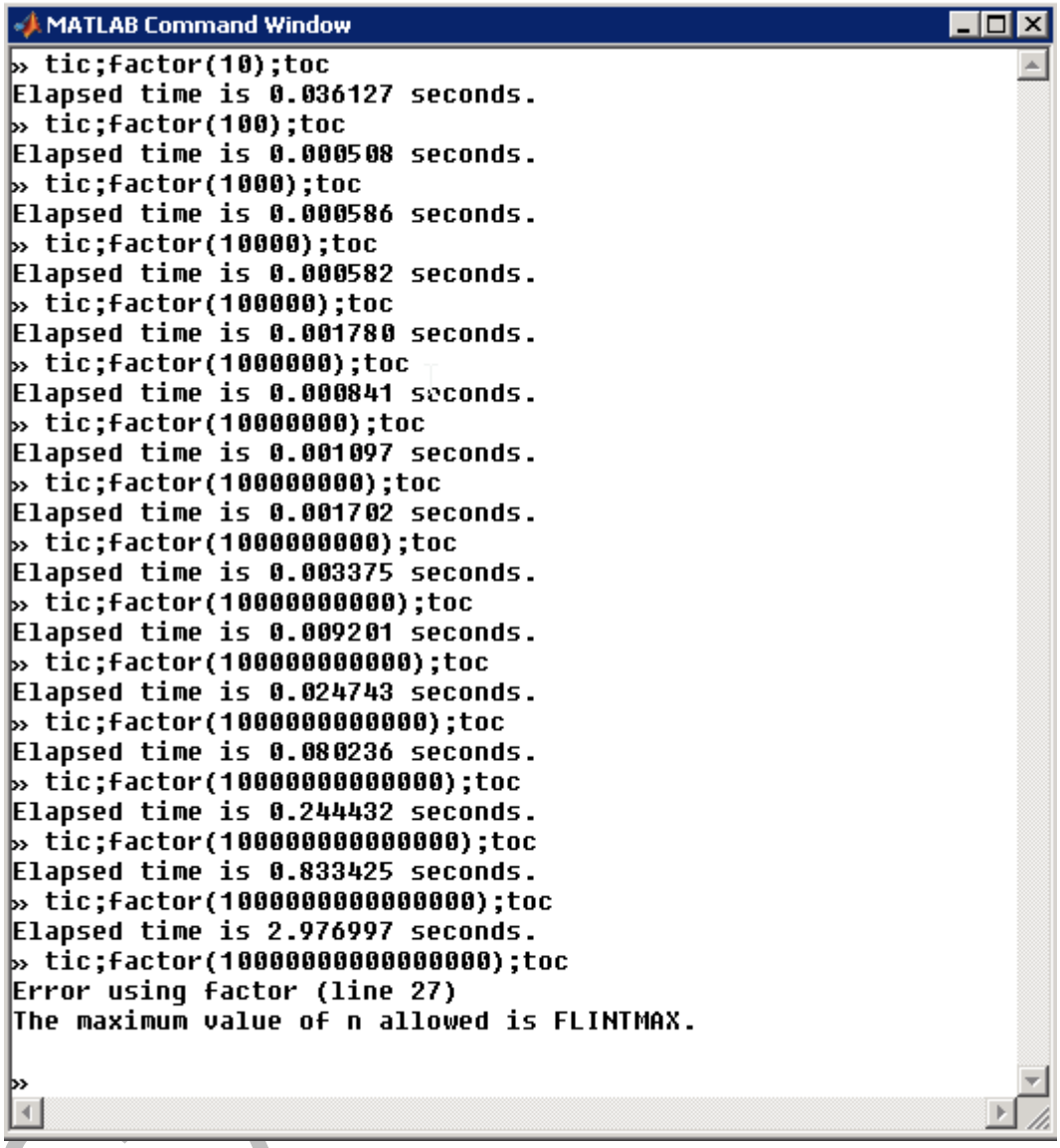
If you need it you can generate small textual calendar in the *Command Window* using the function **calendar( )**:



There is also an important function to add a given time unit to a given date using the function **addtodate( )**:



A famous function when you want to test the efficiency of you code is to use the **tic** and **toc** commands:

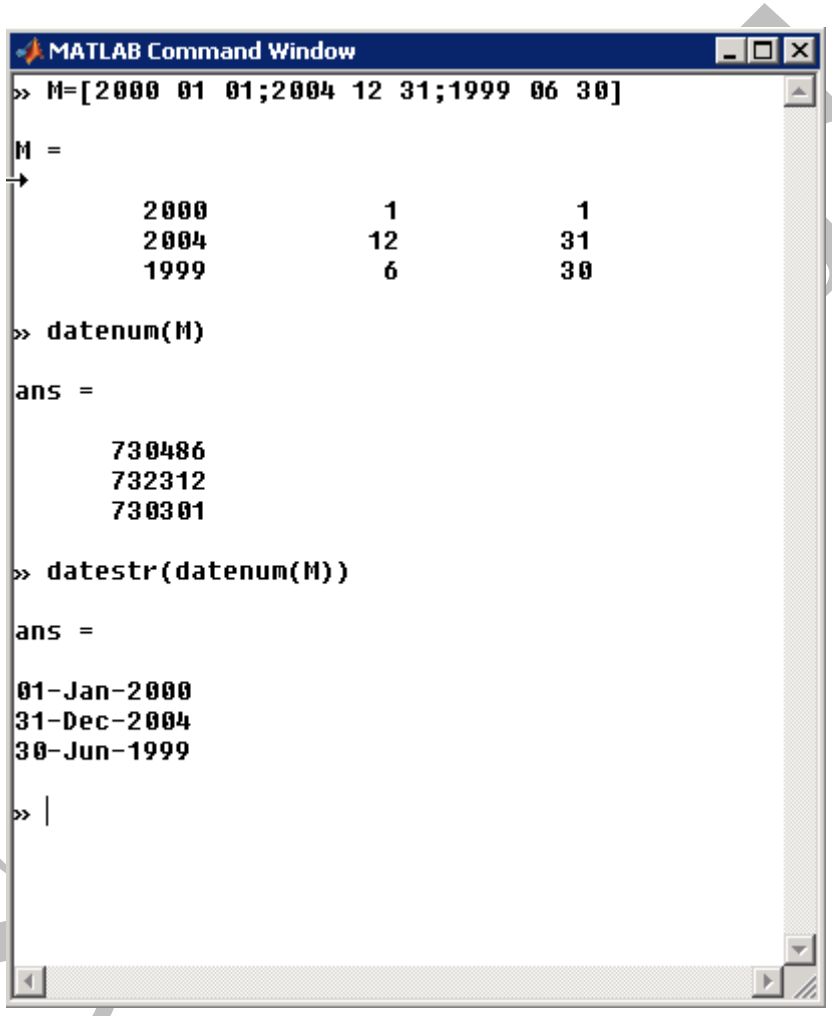


Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seems to have built-in functions.

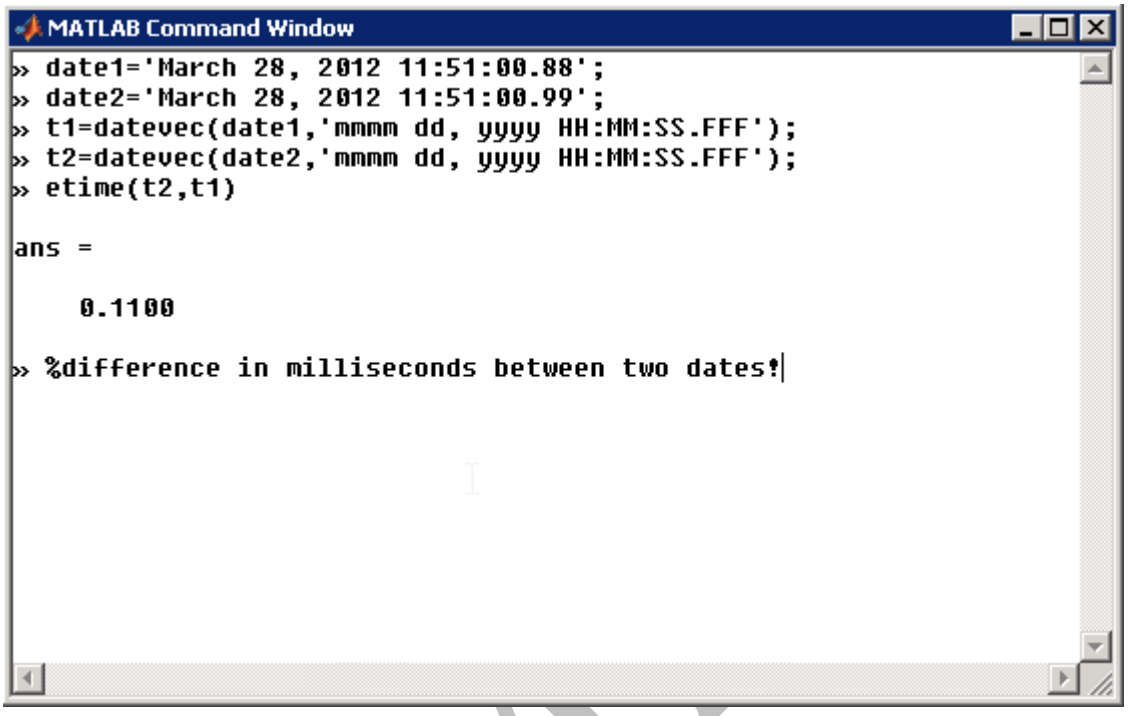
Convert year/month/day matrix as dates

Suppose we have a matrix with 3 columns, each column contains respectively years, months and days. To convert this into standard dates it is in fact very easy. We just have to use the function **datenum( )** and **datestr( )**:

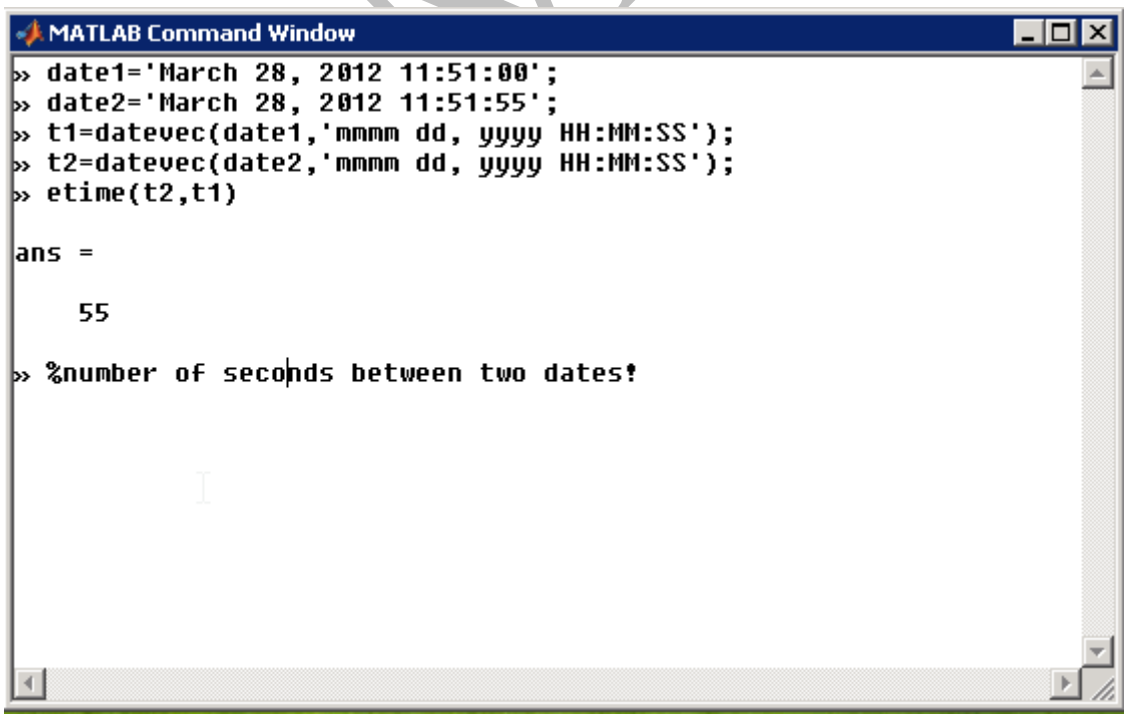


Difference in milliseconds/seconds between 2 dates

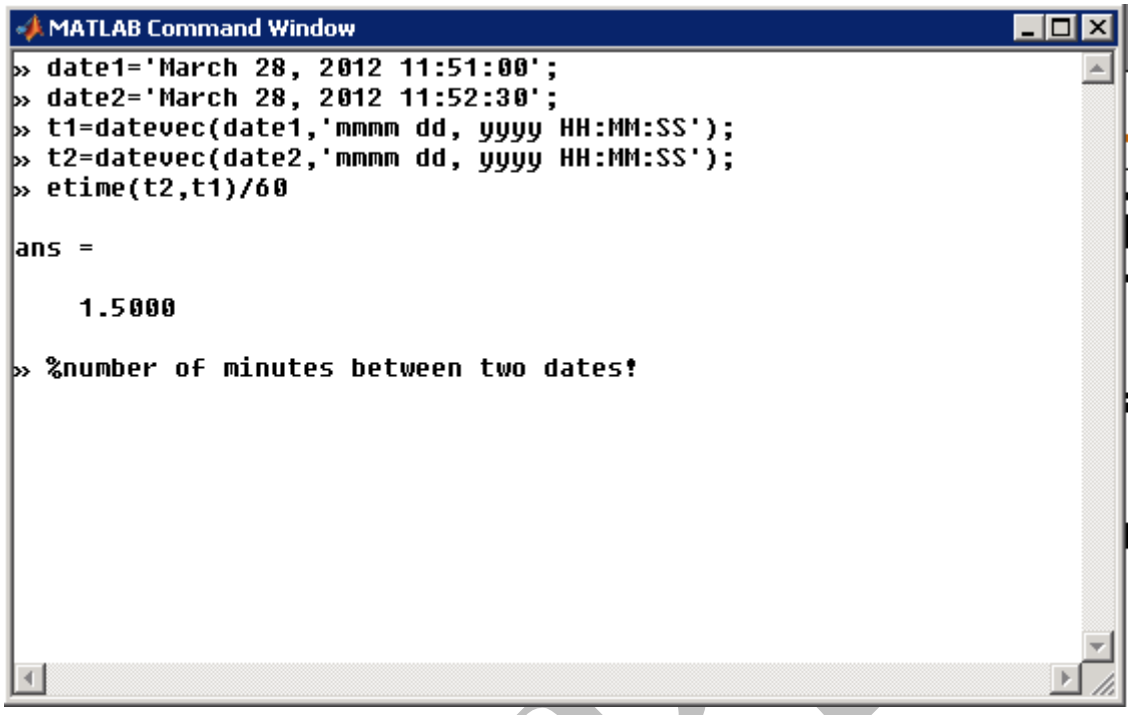
In high frequency trading we need to calculate the number of milliseconds between two times. Supposing that we have two dates in a non-normalized format, we want to calculate the milliseconds between the both times. For this we will use the functions **etime( )**:



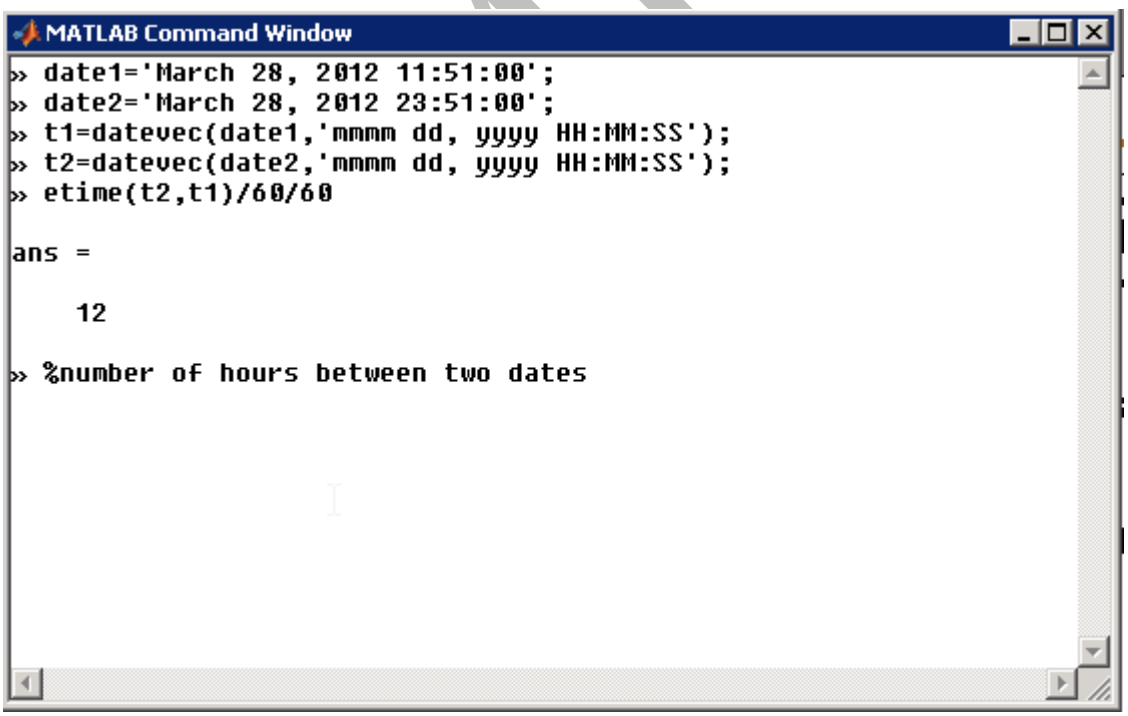
Or if we are just interested to the number of seconds:



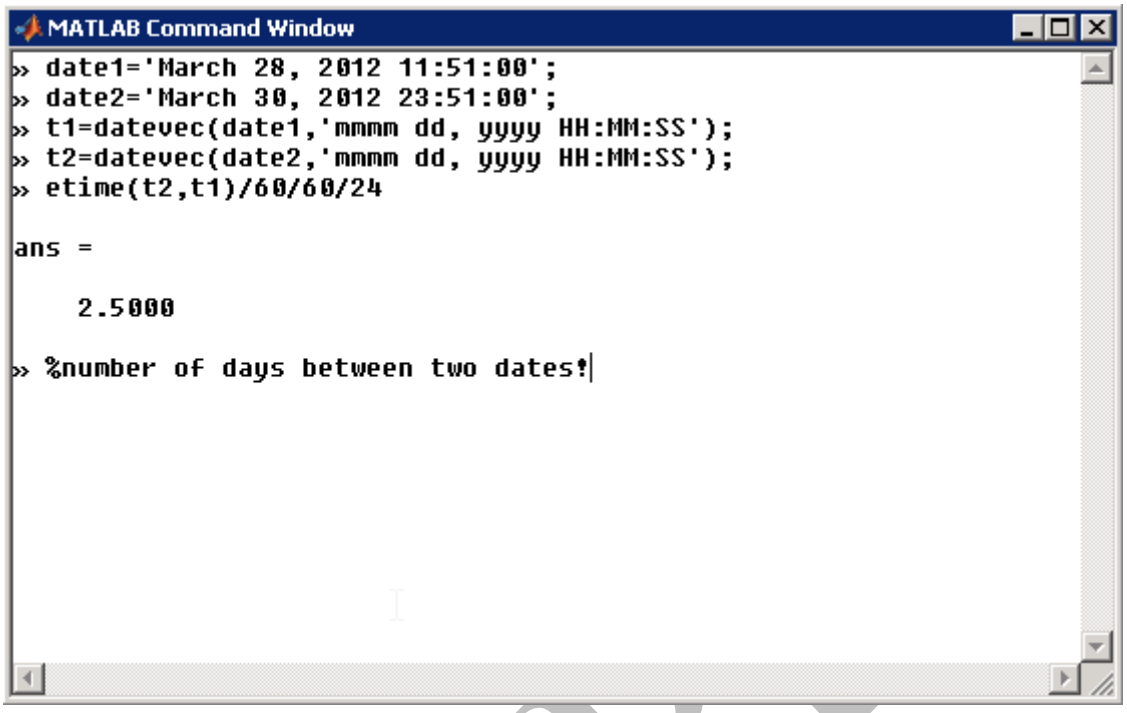
or to the number of minutes by just dividing by 60...:



Or the number of hours by dividing once again by 60...:



Or the number of day by dividing by 24 the number of hours...:

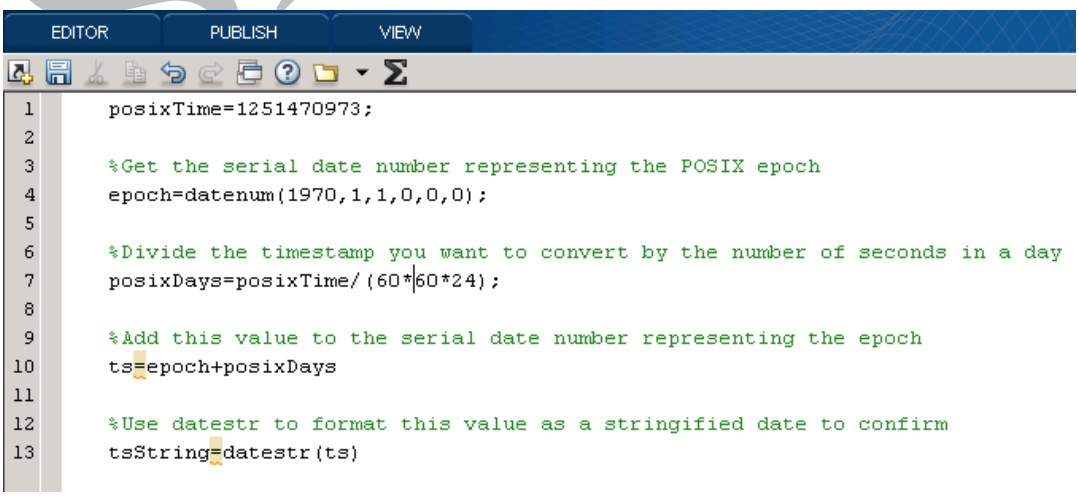


Now for months and years it begins to be a little bit more complicates without Toolbox or scripting...

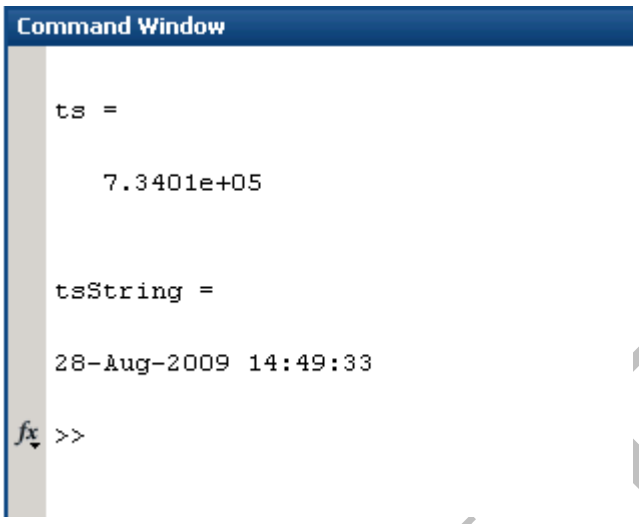
Converting POSIX Time to MATLAB's datenum format

POSIX time (also called unix or epoch time) is the number of seconds which have elapsed since January 1, 1970 GMT, which is referred to as the 'epoch'. The main advantage of this unit of time is that it represents the same instance in time around the globe, regardless of time zone.

MATLAB uses serial date numbers to represent time. A MATLAB serial date number is defined as the number of days (and fraction thereof) which have elapsed since January 1, 0000. Unlike POSIX times, this value is not absolute; ie: these timestamps represent different moments in time, depending on the time zone.



We get as result the correct one (you can control that easily with any worksheet software):



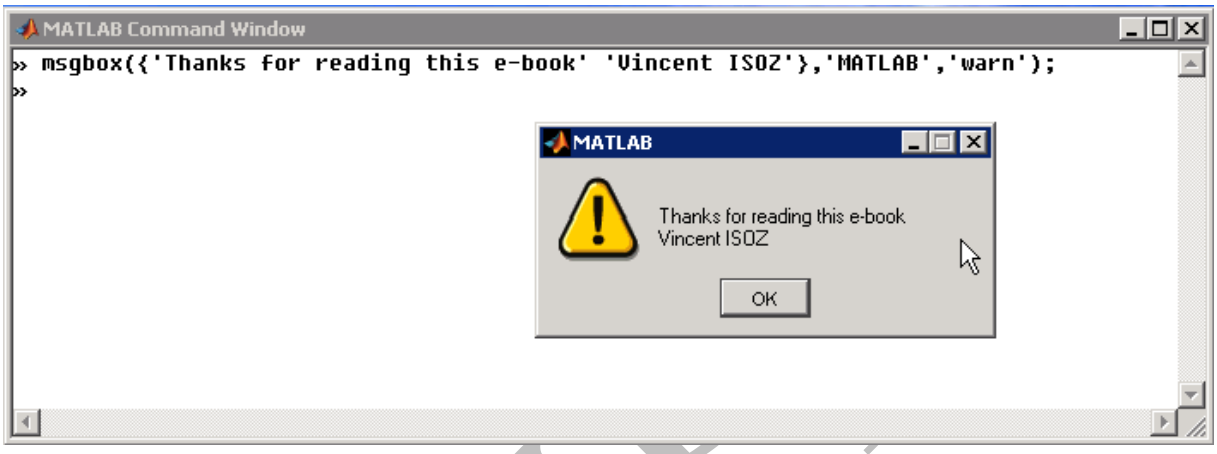
SAMPLE

Simple user interactions

Message boxes

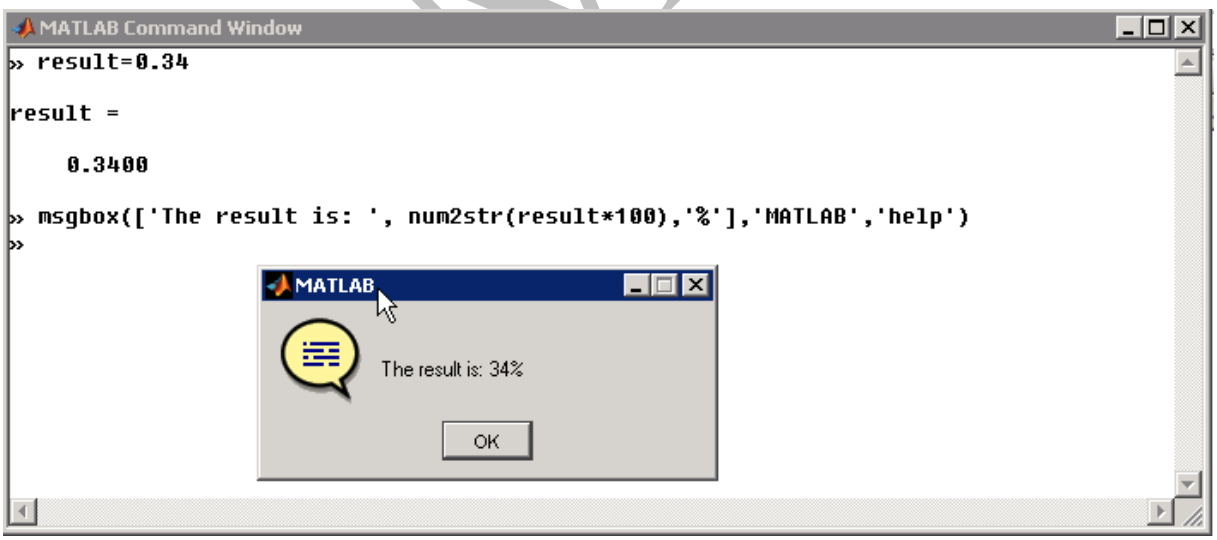
Later when we will learn to write scripts it will be necessary to interact with the user. The most common elementary function for this purpose is to use the function **msgbox( )** that has three different logos (*warn*, *help*, *error*).

Here is a demo first withe the warn logo:



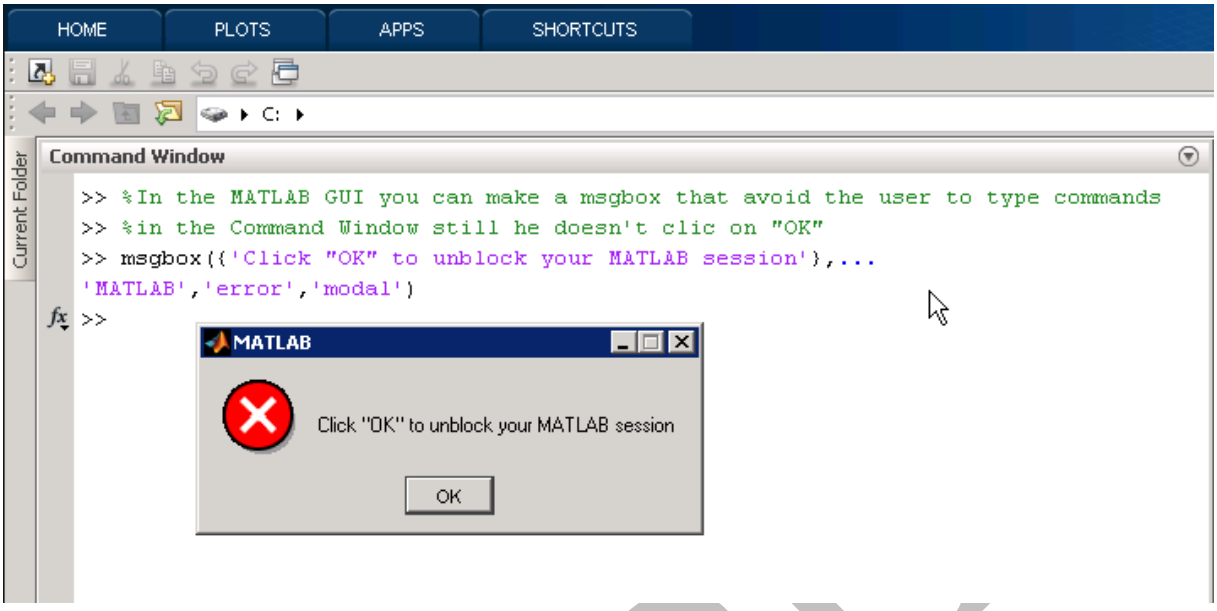
Remark: It is possible to customize the icon with an external image but at this step it is not a good idea because **the image won't follow** your script if you don't create a package (toolbox).

Now an example with the help logo and the result of a calculation:



Remark: You can't add nice equations in a message box without the Symbolic Toolbox...

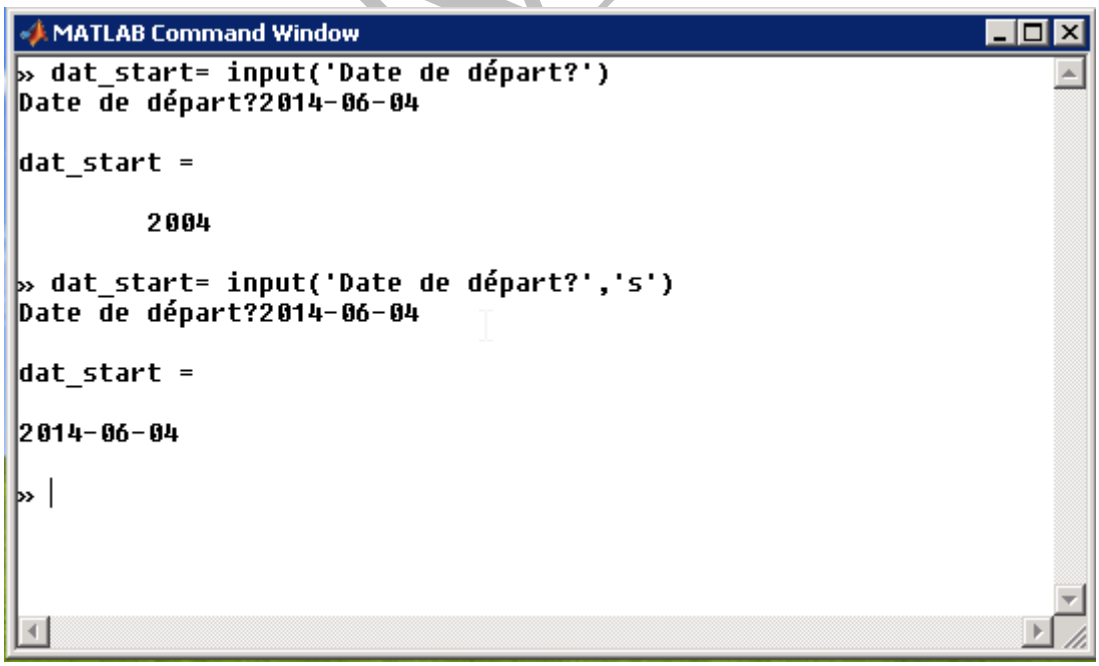
The user can still write behind the message box in the leight Command Window. To block the user make the message *modal* as following but **it works only in the MATLAB GUI...**:



All others user interactions that we will se below (inputdlg, listdlg, etc.) are "modal" in the MATLAB GUI by default!

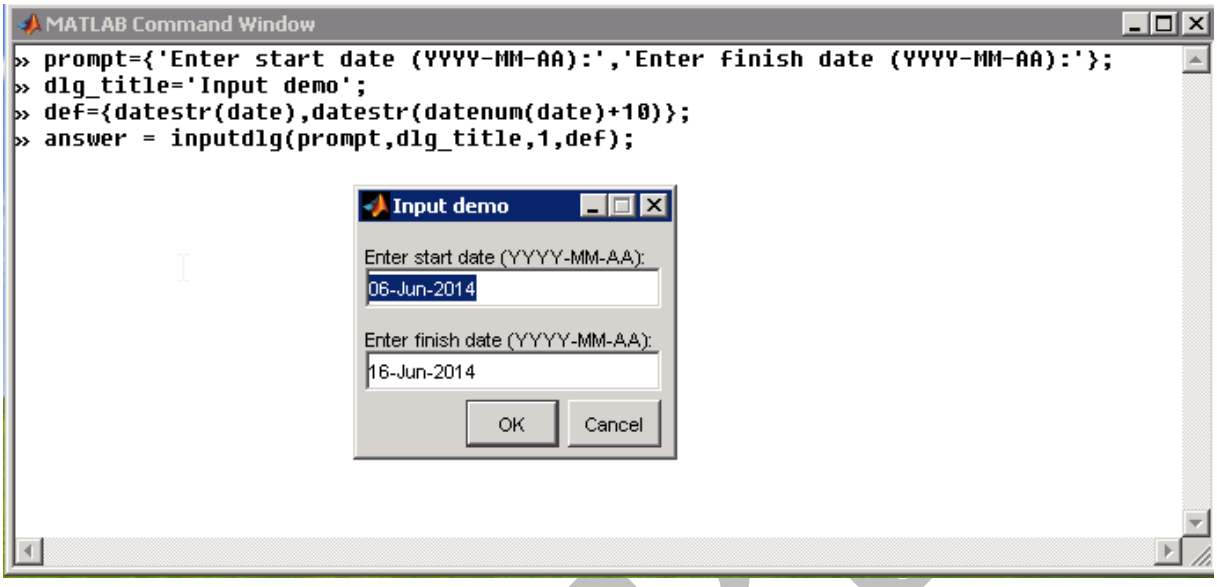
Input

Now let us see a very basic way to ask the user for an input using the Command Windows and the function **input( )**. See also the trap specific to dates that you have to define as string input:

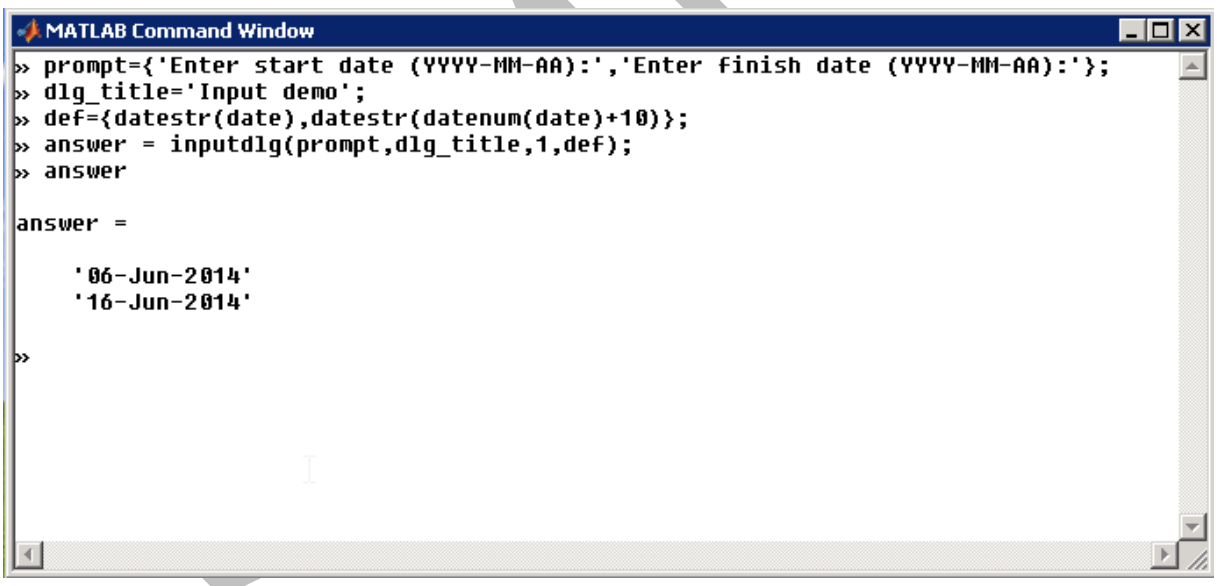


Input Dialog

The previous **input( )** function needs the *Command Windows* to be open. This is not a good thing. A much more interesting alternative is to use the **inputdlg( )** function:



And here you can see the result of the input:

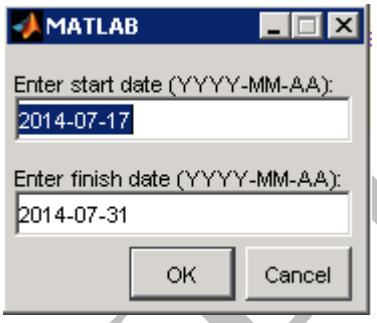


But it's still not right (see the dates! the format does not correspond). We can correct this using our knowledge on dates functions:

```
prompt={'Enter start date (YYYY-MM-AA):','Enter finish date (YYYY-MM-AA):'};
def={datestr(date,'yyyy-mm-dd'),...
    datestr(addtodate(now,eomday(year(date),month(date))-day(date),'day'),'yyyy-mm-dd')};
answer=inputdlg(prompt,'MATLAB',1,def);

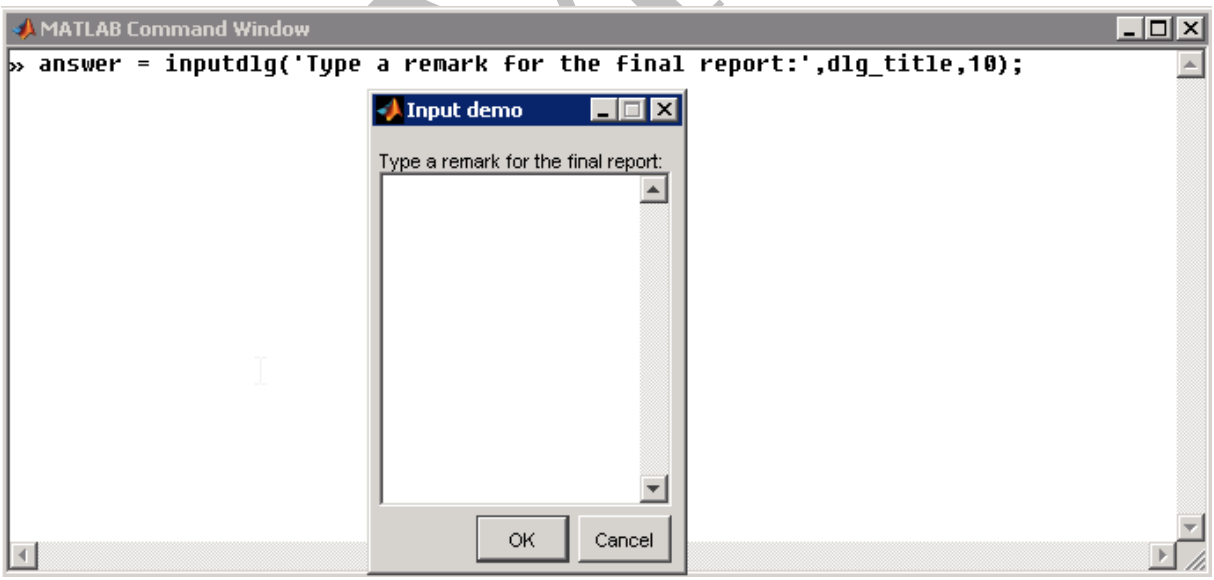
```

Then we get:

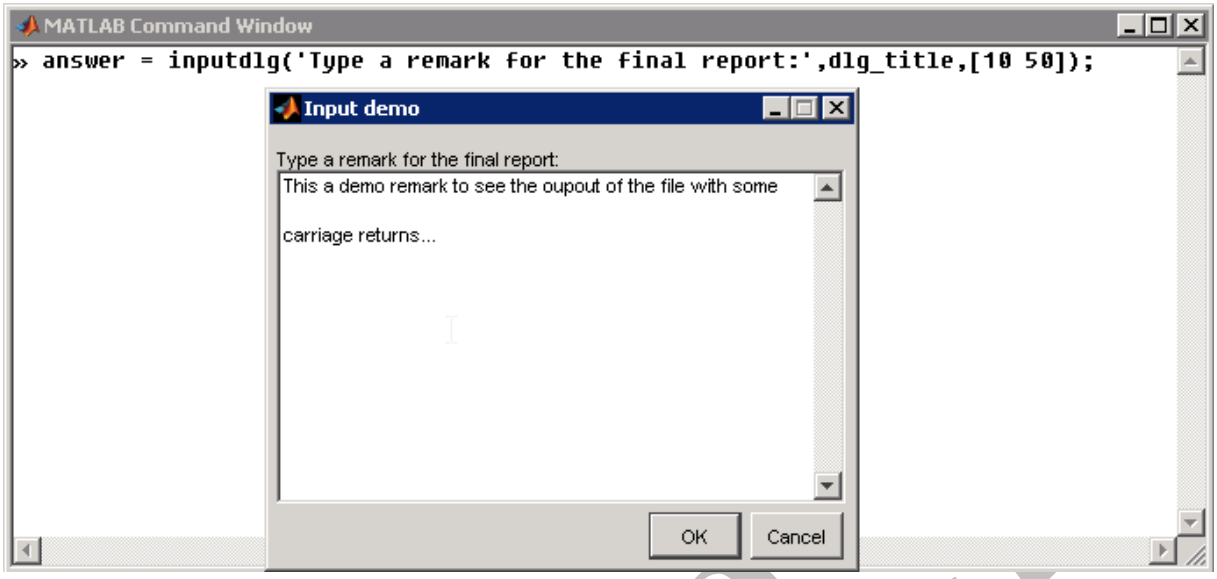


That's it!

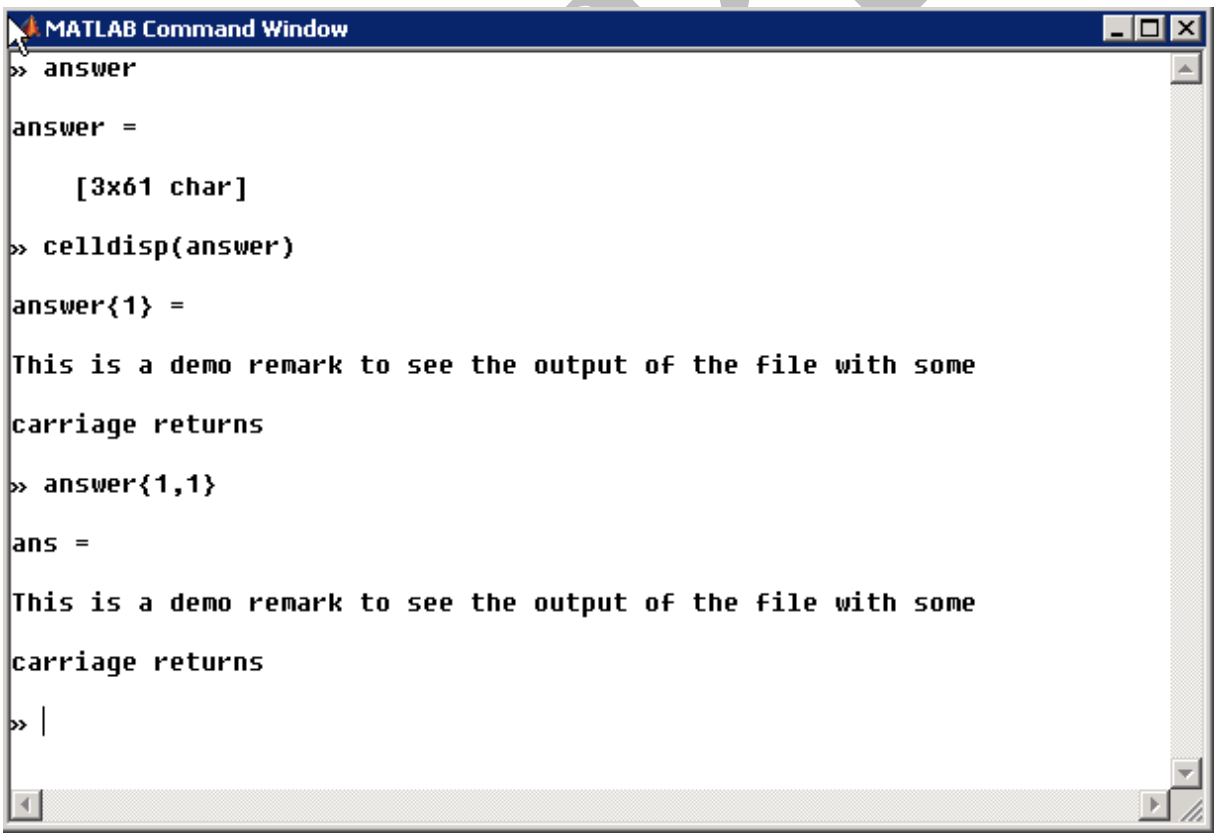
Let us see what is the purpose of the last argument of the **inputdlg( )** function:



Or we can also control the width of the **inputdlg** field:

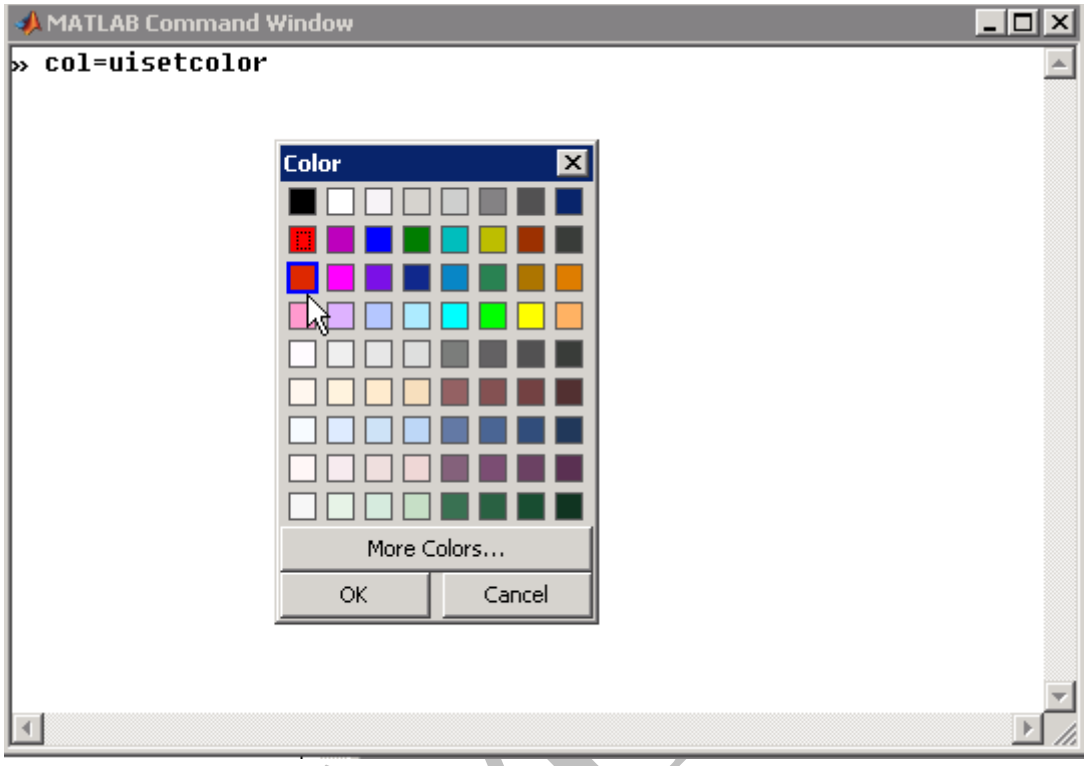


As you can see the input written by the user will be in a cell array we can see the content of this cell array by using a classic technic or by using the **celldisp( )** function:

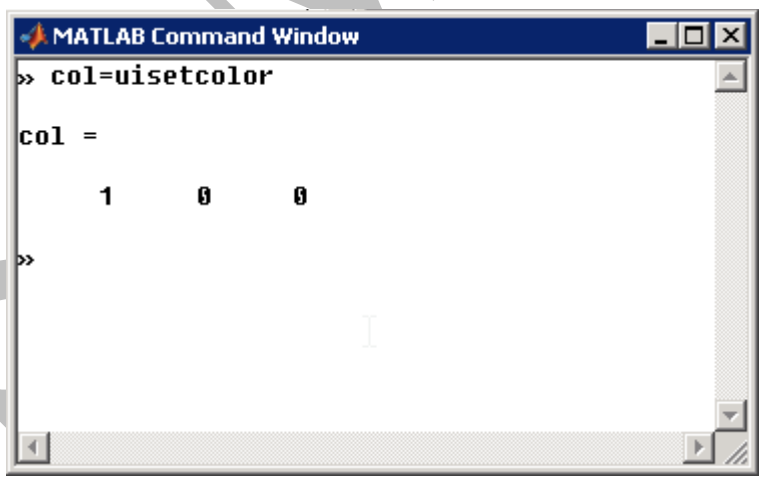


UI set color

Another very useful user interaction is the **uisetcolor** command that gives the possibility to the user to choose a color when you build automated charts:



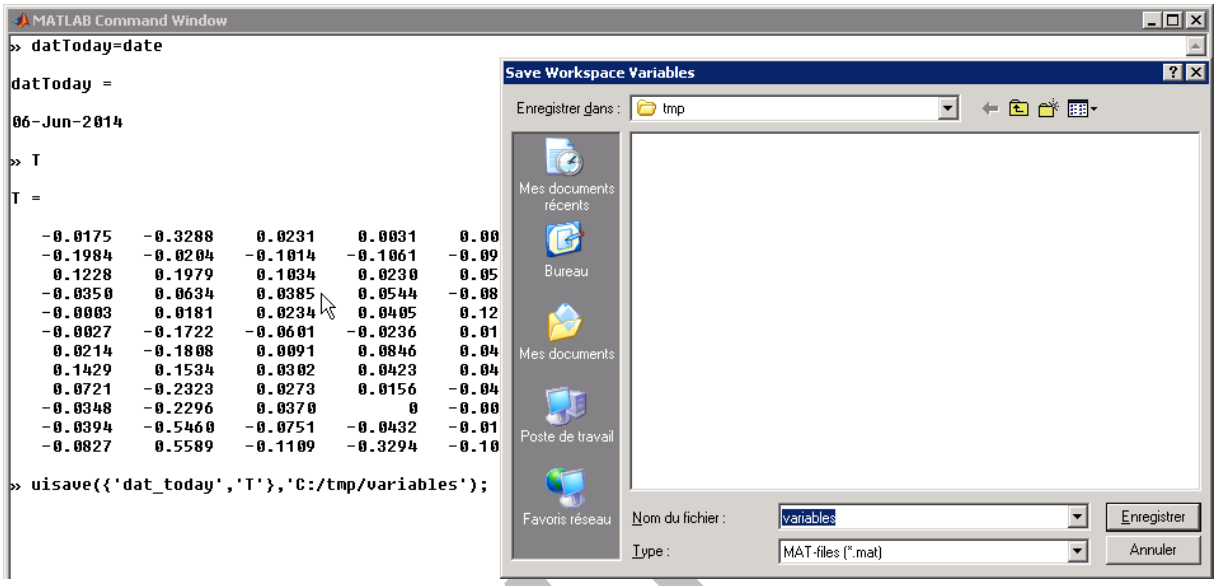
When the user choose a color the result will be the following:



UI save

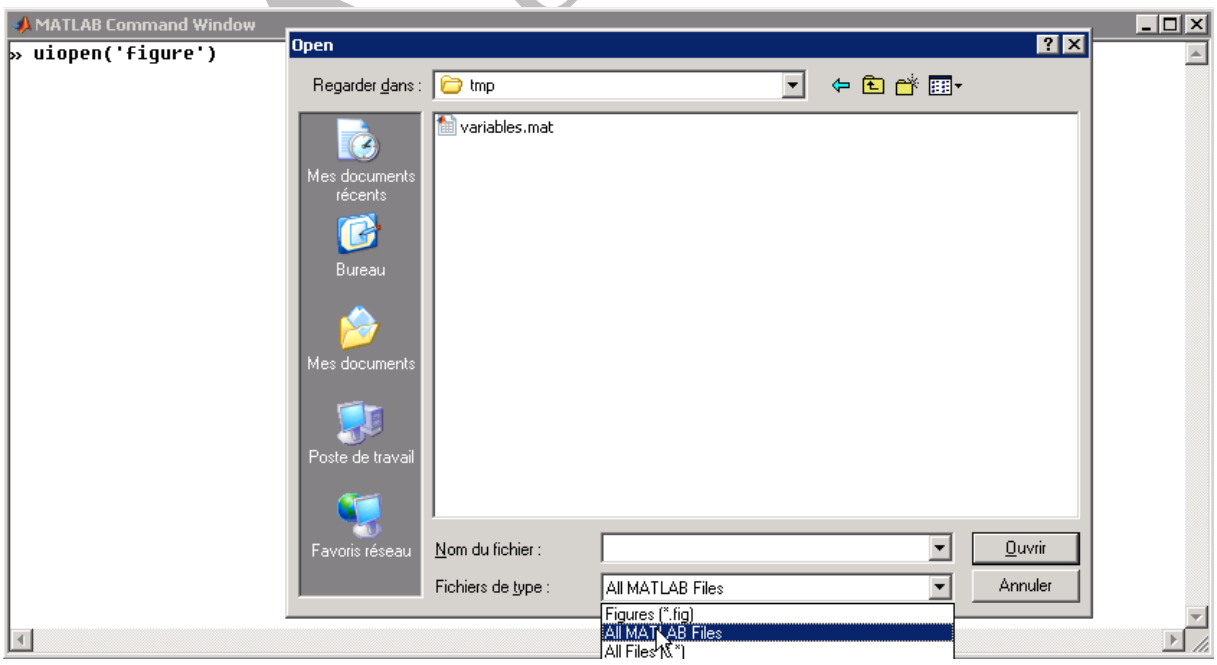
As for the previous interactions options you will also need with some scripts to save all variables of the current MATLAB session in a *.mat file.

For this we can use the **uisave( )** function:

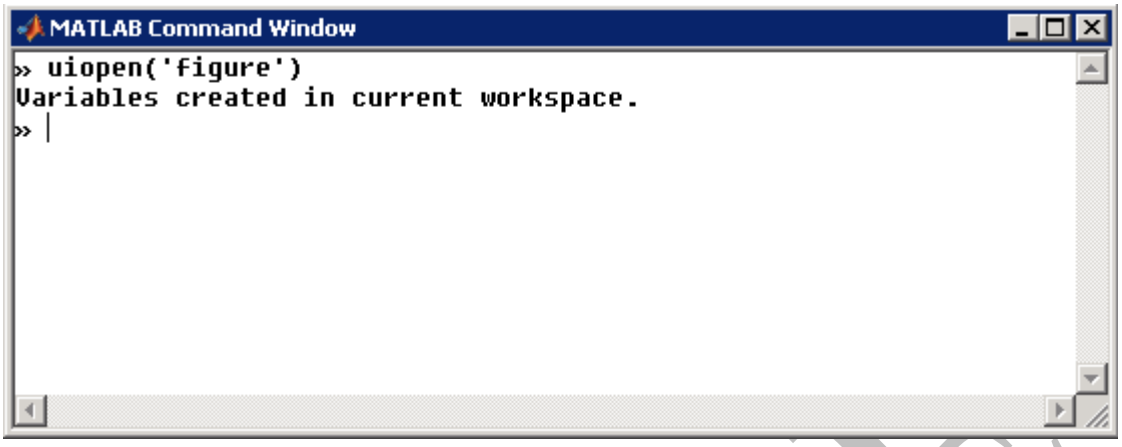


UI open

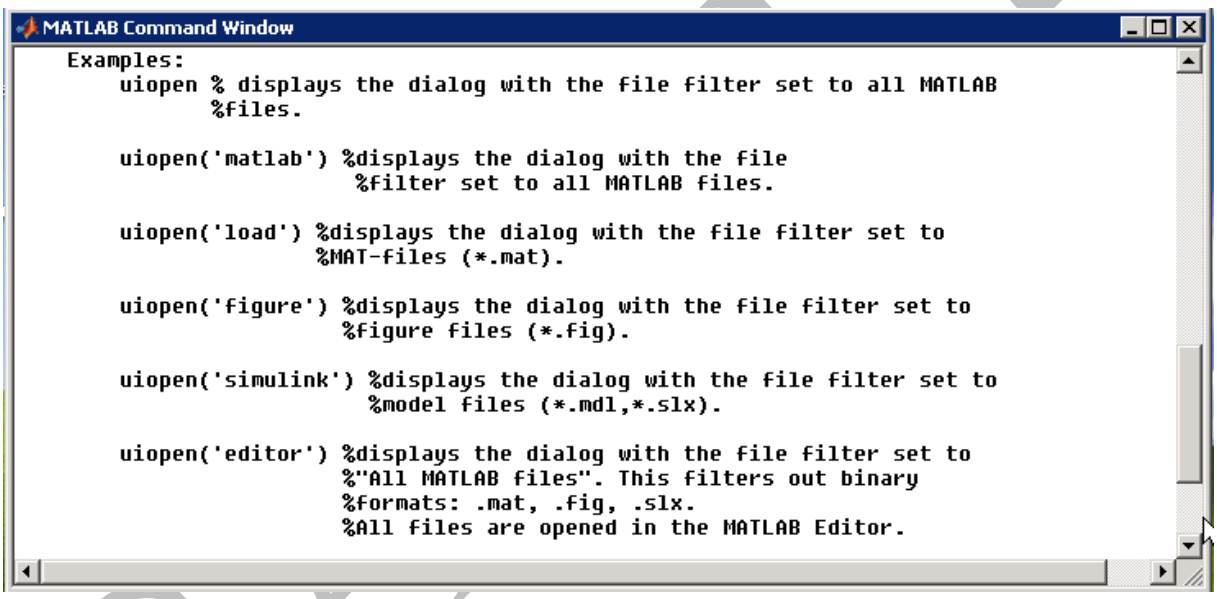
It can also be sometimes useful to ask the user interactively which *.mat or *.fig he wants to open with the command **uiopen( )**. The 'figure' doesn't mean we want the user to be able to open only figures but just that MATLAB show us the dialog window to open files with by default the *.fig activated:



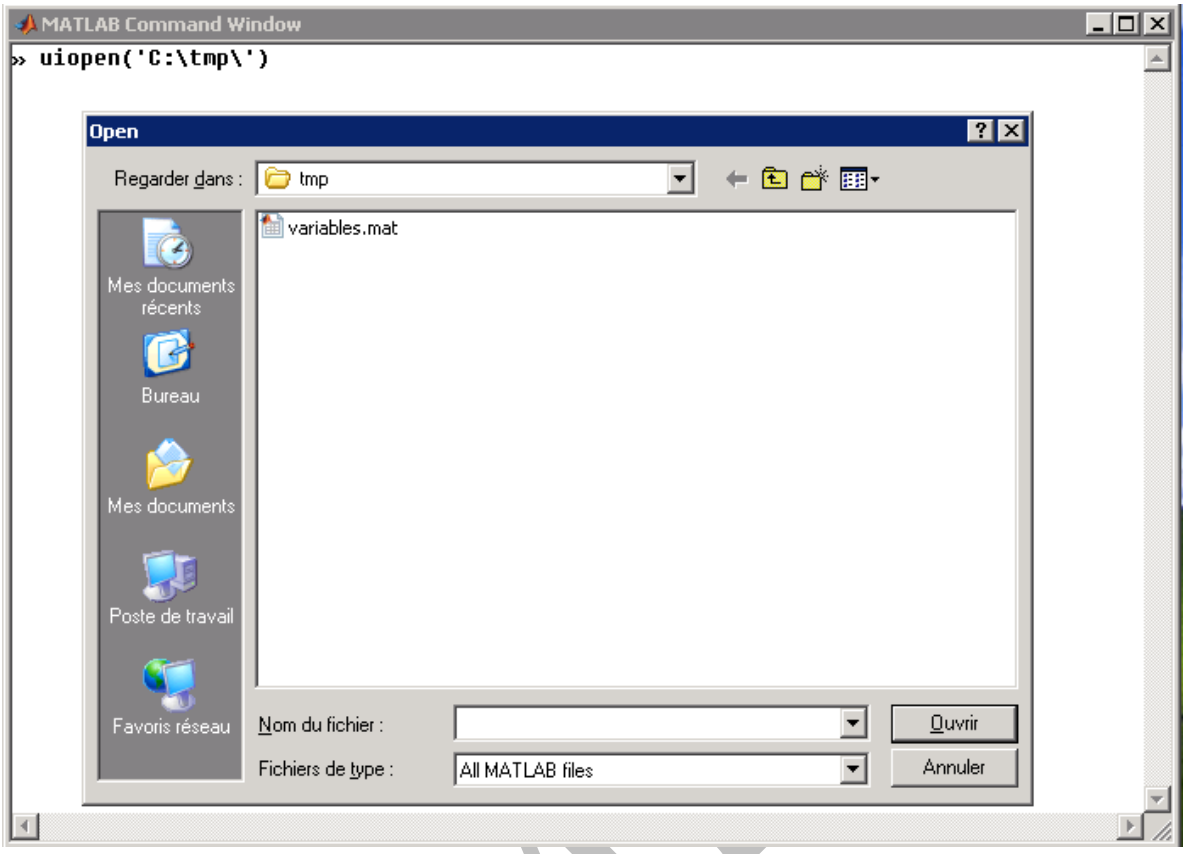
If we select a *.mat file instead of a *.fig it will work:



The **uiopen( )** function has also other arguments to filter the visible filetypes. As you can see in the help of the command:

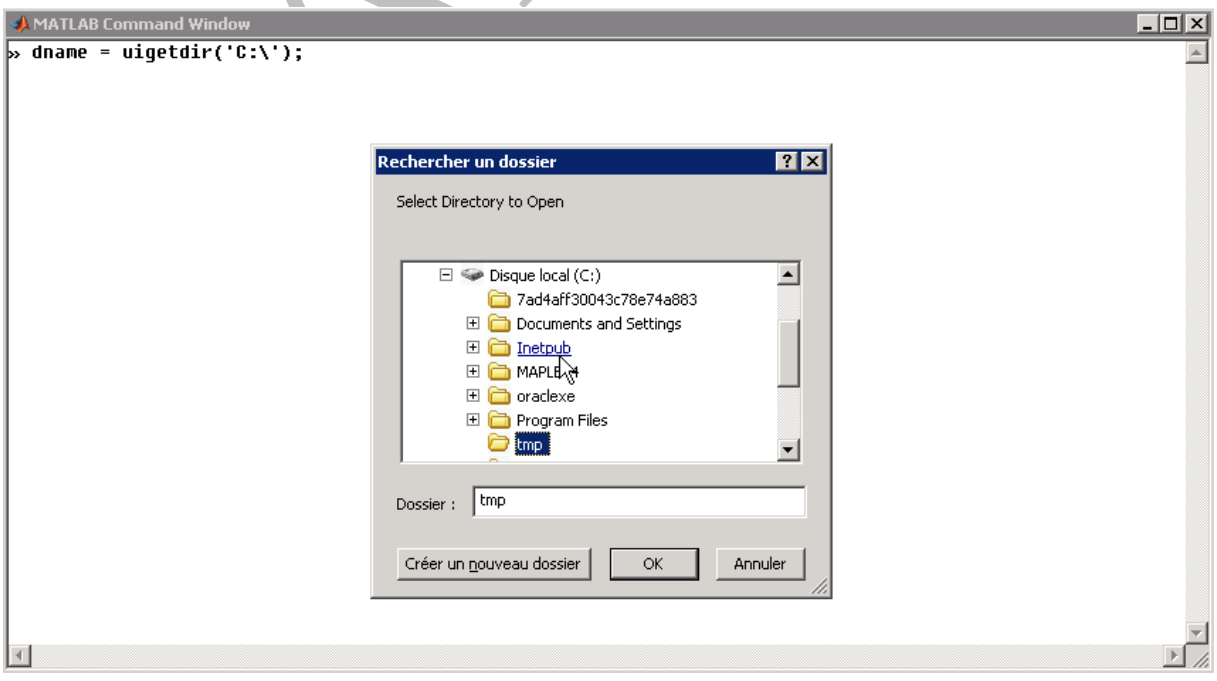


We can also force the open dialog window to show a given folder by default:

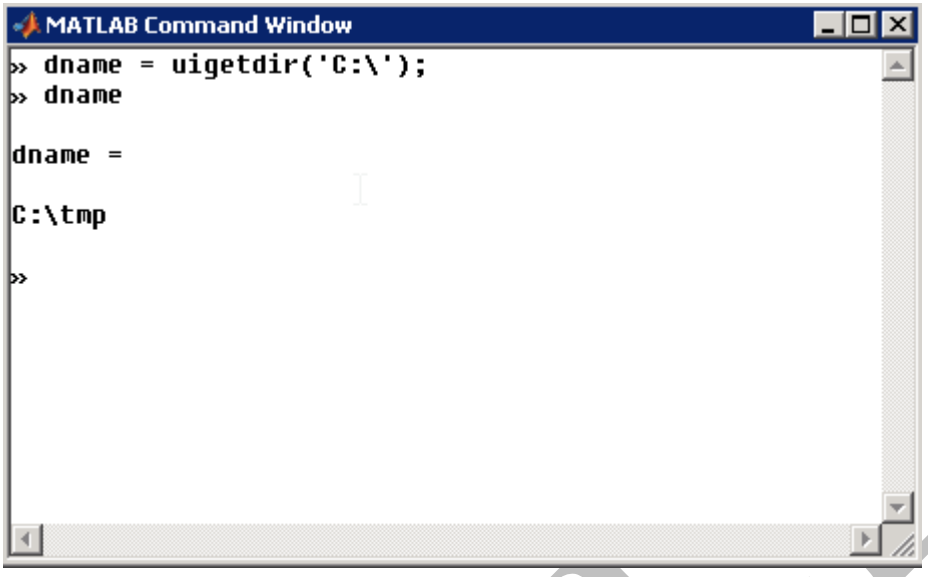


UI get dir

The need to as the user a target director with the function **uigetdir( )** is something very common when you develop MATLAB applications to ask the user where he wants to store the reports:



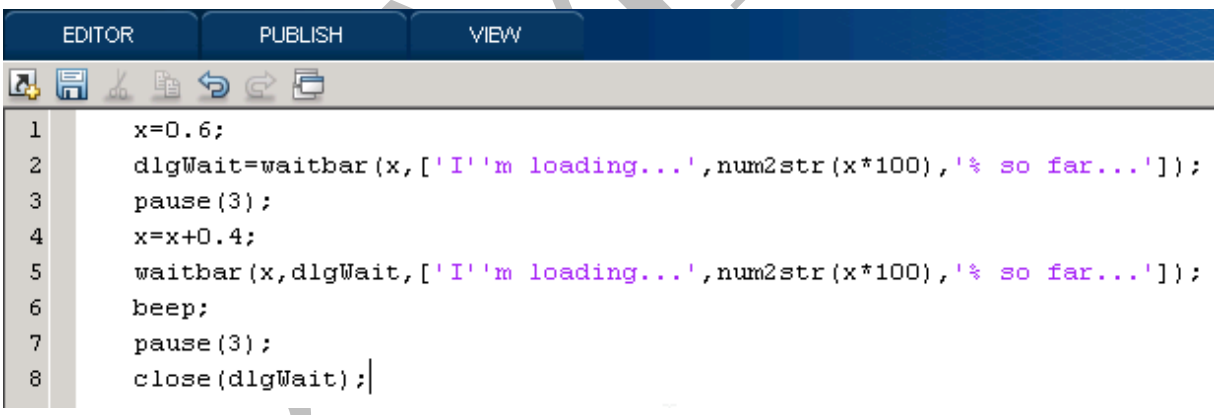
As you can see the result will be a simple string:



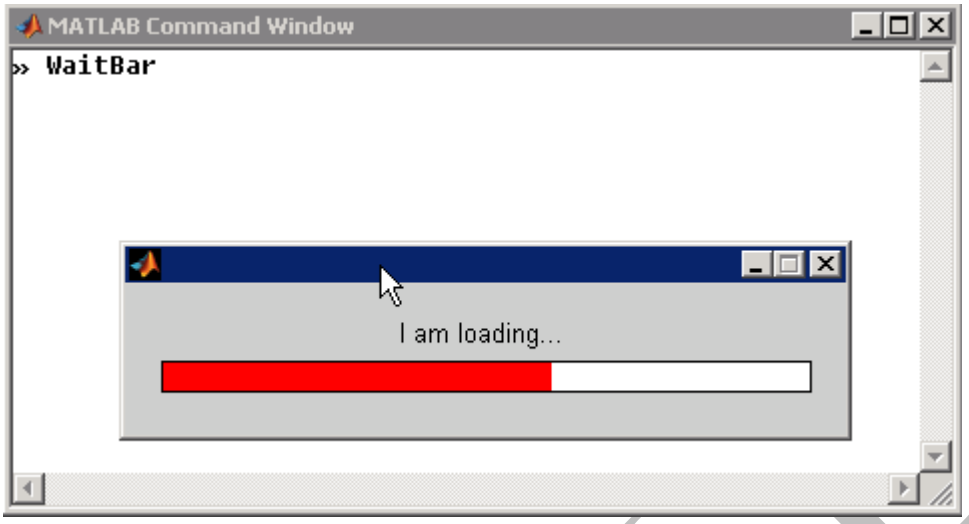
Wait bar

A functionality that is request a lot by MATLAB users but slow down scripts a lot is to show a wait bar to the user of you application.

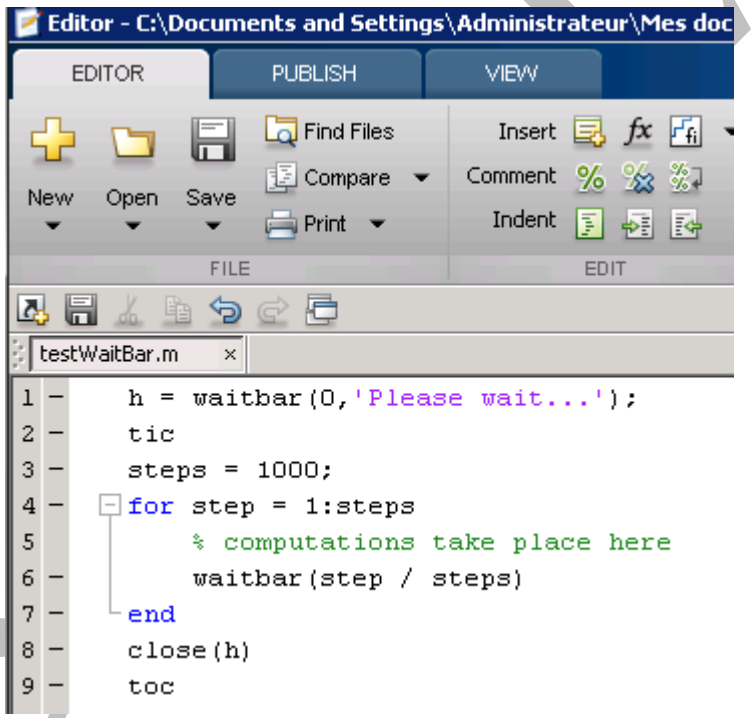
Below you can see a very simple stupid usage of the **waitbar( )** function that starts at 60% then after 3 seconds jump to 60%+40% and do a **beep** and after close:



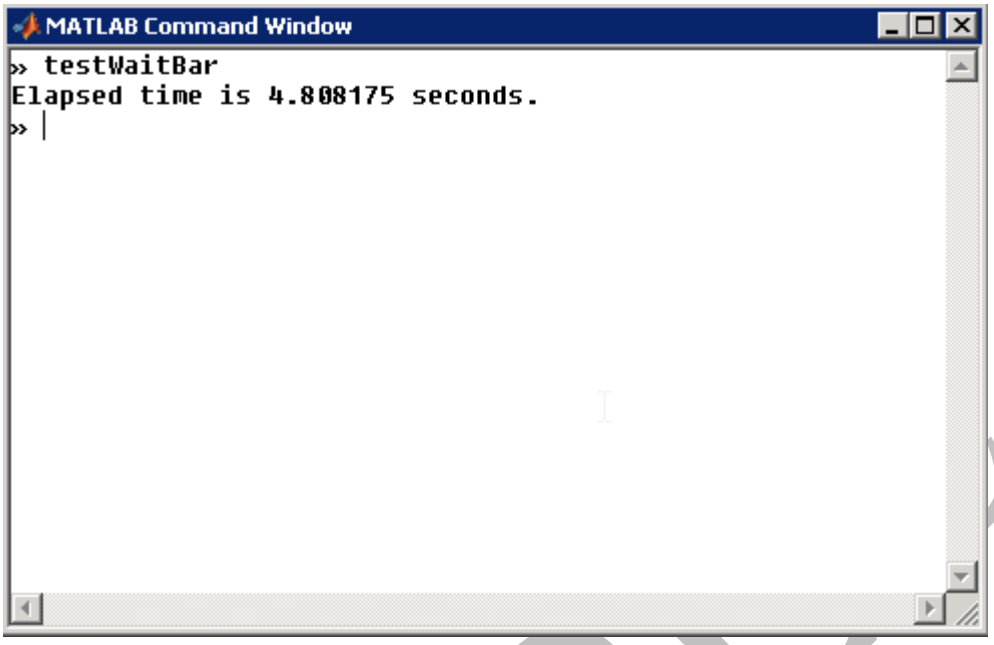
The wait bar will look like this:



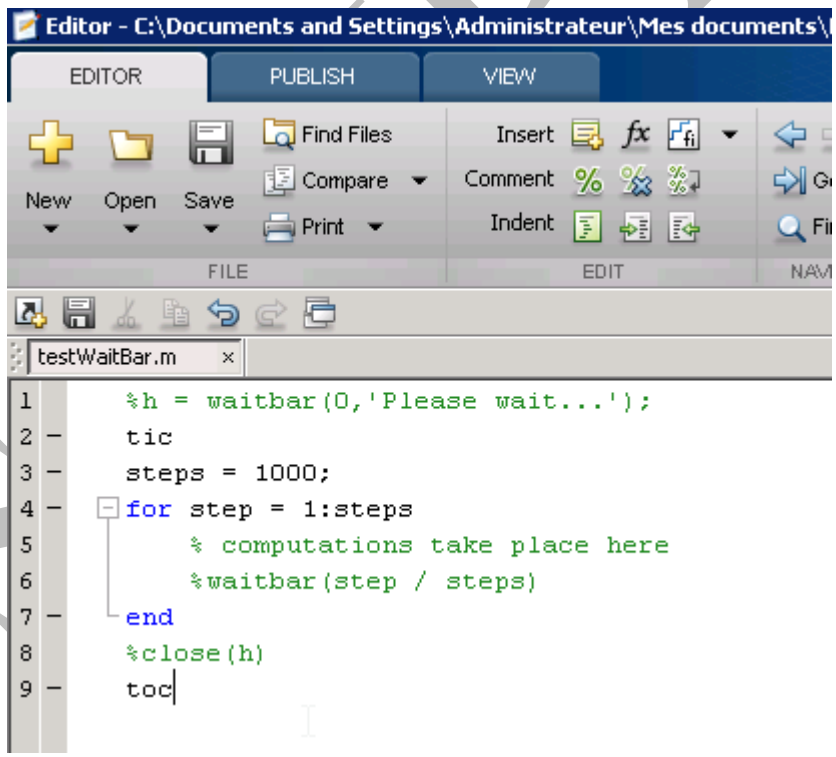
And here is a more real typical example of the use of the wait bar in a loop:



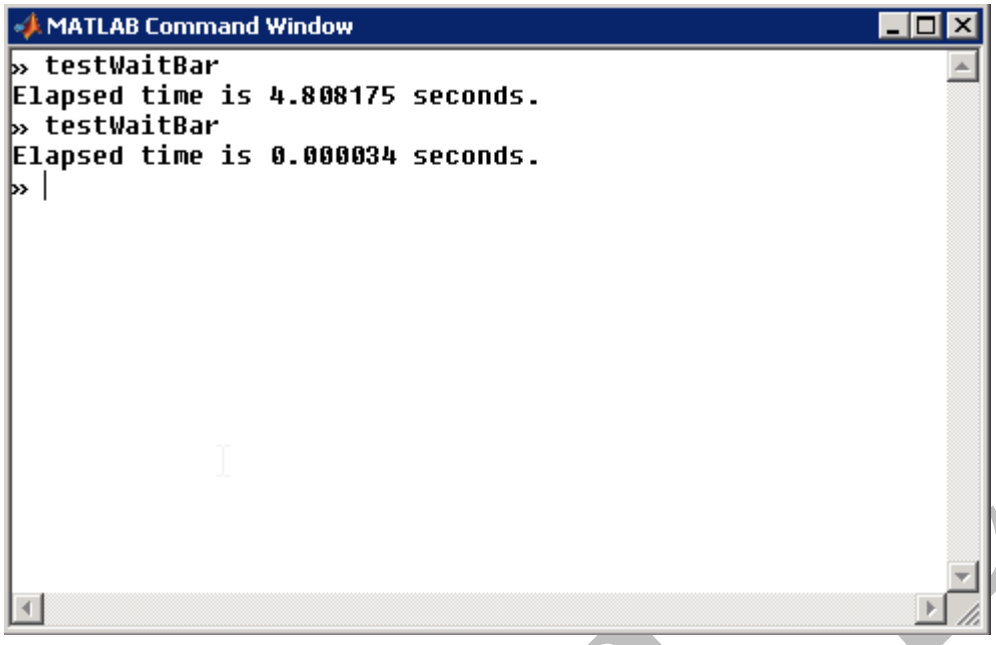
And here you can see the execution time with a wait bar:



And now let us deactivate the wait bar:



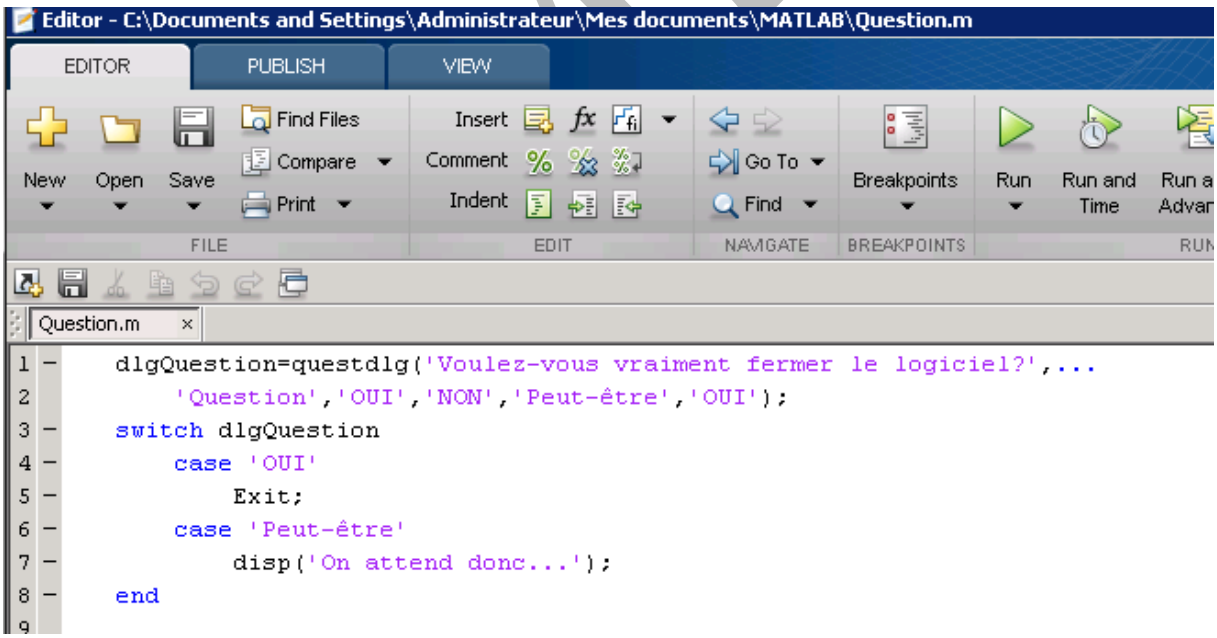
and see the execution time...:



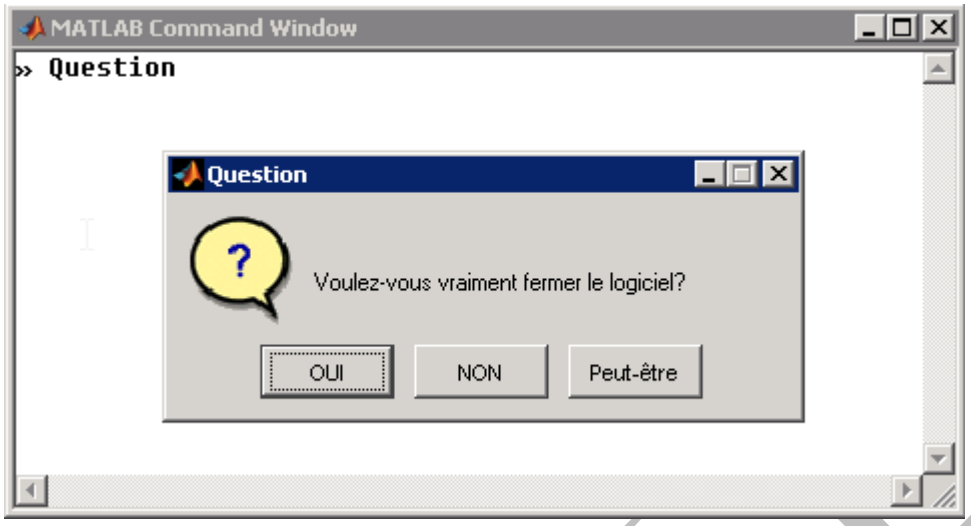
no comment...

Question dialog

Another typical dialog is to ask a question to the user and retrieve its answer. This can easily be done thanks to the function **questdlg( )**:



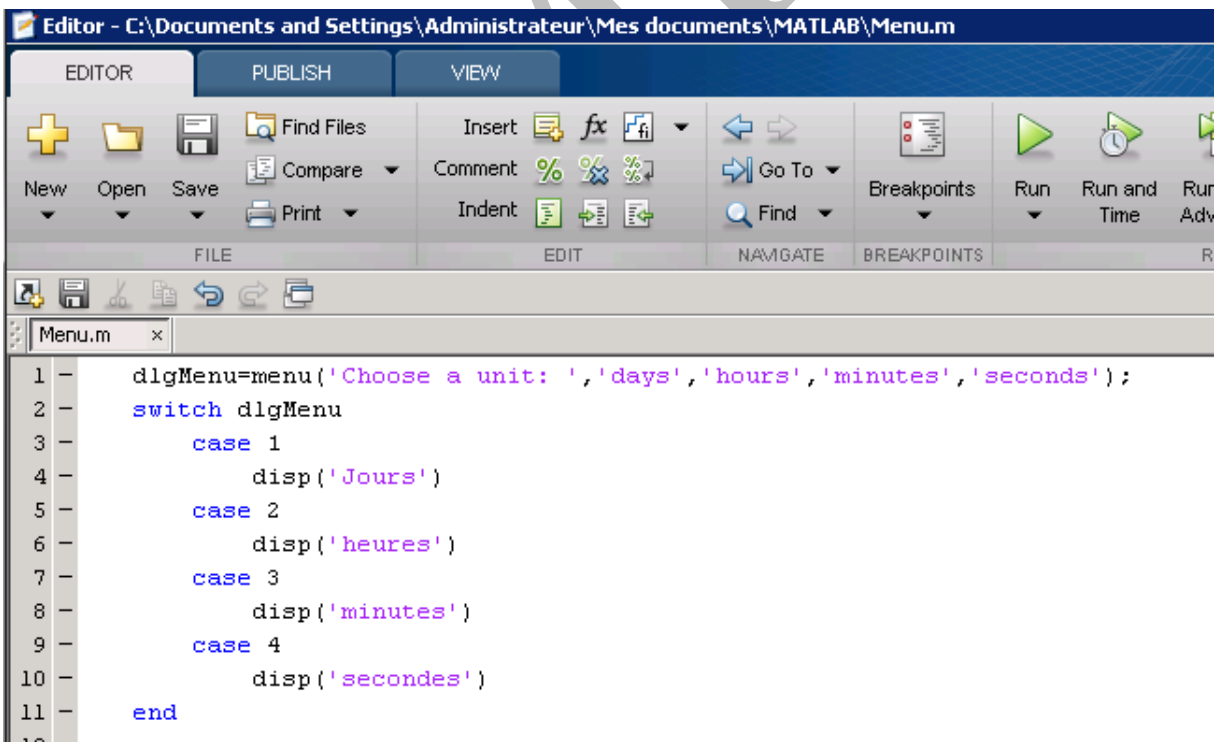
This will result in:



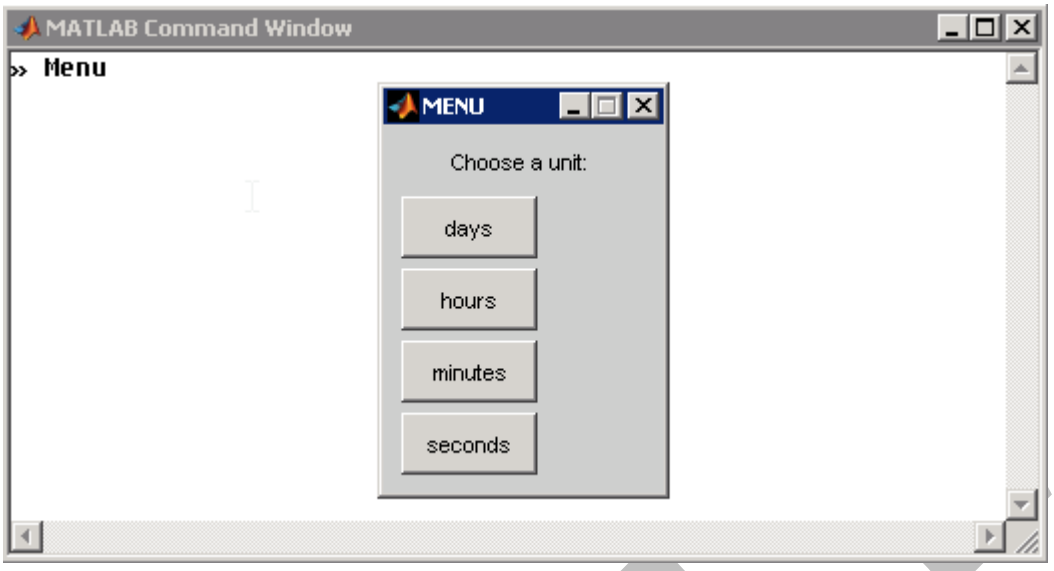
and the answer will be automatically managed by the *switch*... *case*.... structure!

Menu

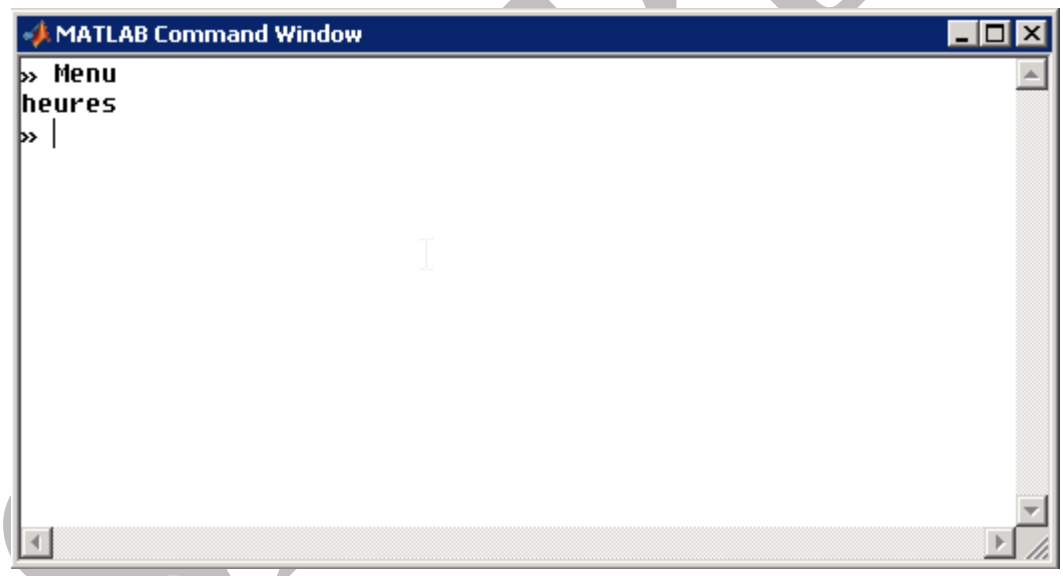
When you make charts it is many times useful to ask the users the unit scale he wants to see for the axes or the export format for automated reports. For this purpose one possibility is to create a menu using the function... **menu( )**:



This will give:

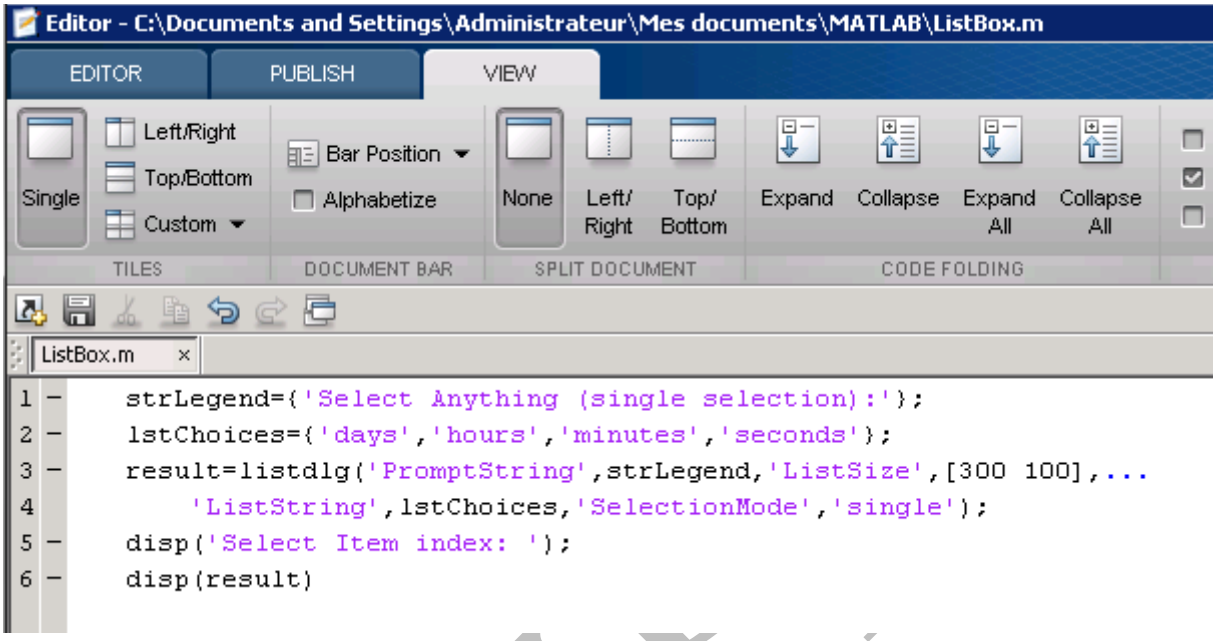


and the answer will be automatically managed by the *switch*... *case*.... structure as you can see it below:

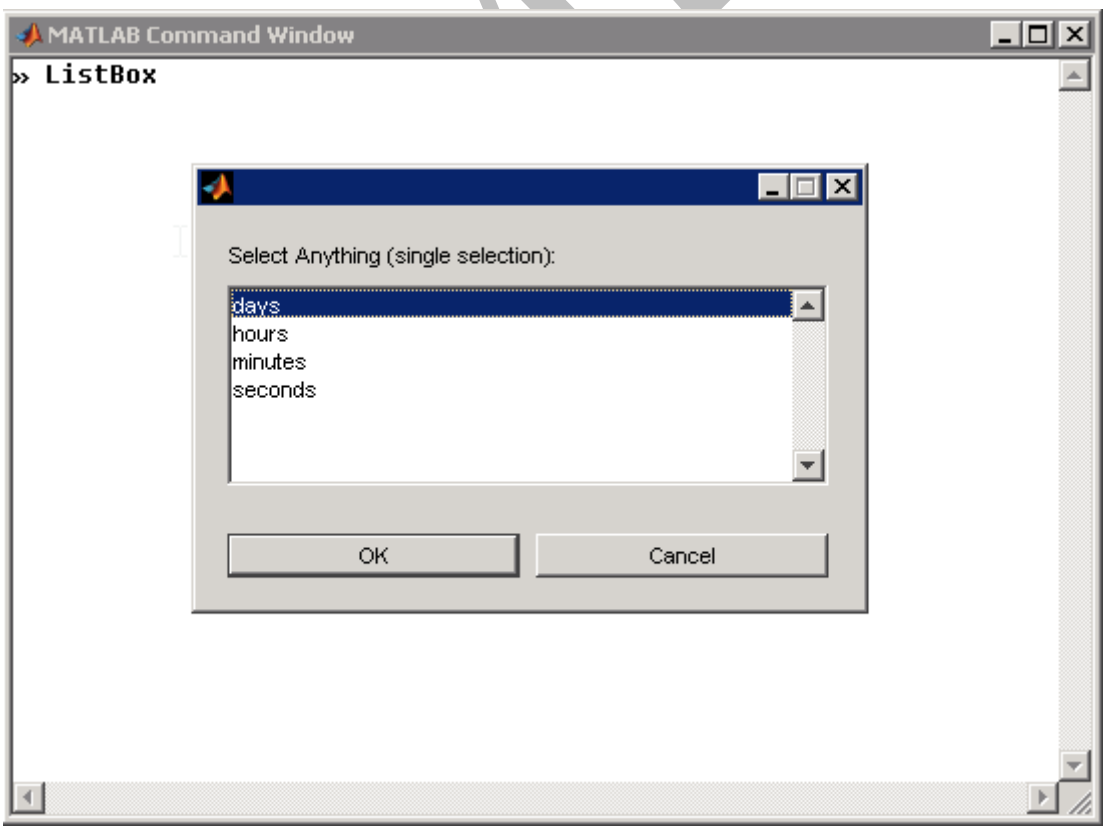


List dialog

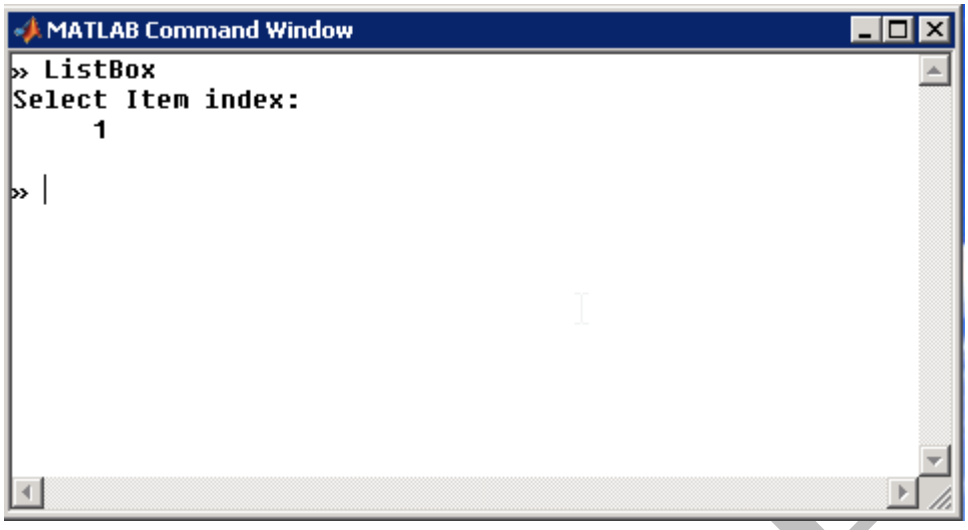
An alternative to the menu is to create a list dialog using the function **listdlg( )** but this last option has something more... the user can make multiples choices if you want to let him do it.



The result will look like this:



And as for the menu, this function returns the choosen index numbers:



Send E-mail

Another very common request is to send automatically e-mail trough MATLAB to warn financial portfolio manager of some events. Here is a typical script that you can use and customize depending on your needs that is based on the **sendmail( )** core function:

EDITOR PUBLISH VIEW

```
% Define these variables appropriately:
mail_address = 'isoz@sciences.ch';      % My E-Mail Address
user_name = 'isoz';                     % My Username
password = 'mypassword';                % My E-Mail Password
smtp_server = 'mail.infomaniak.net';    % My SMTP Server

% Then this code will set up the preferences properly:
setpref('Internet', 'E_mail', mail_address);
setpref('Internet', 'SMTP_Username', user_name);
setpref('Internet', 'SMTP_Password', password);
setpref('Internet', 'SMTP_Server', smtp_server);
props = java.lang.System.getProperties;
props.setProperty('mail.smtp.auth', 'true');
props.setProperty('mail.smtp.socketFactory.class', 'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port', '465');

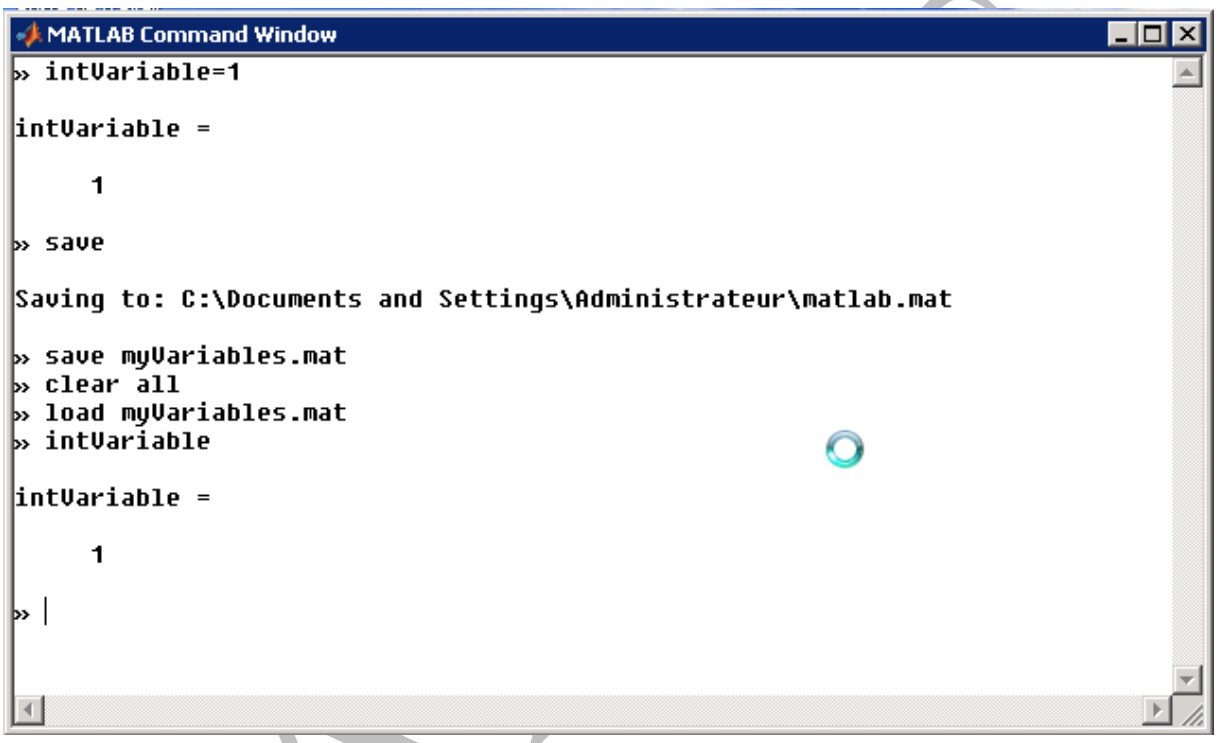
% Send the email with carriage return in body text (10) and attachments
sendmail('mymail@somedomain.com', 'Mail from MATLAB', ...
    ['Line1 of message' 10 'Line2 of message' 10 ...
        'Line3 of message' 10 'Line4 of message'], ...
    {'report.doc', 'chart.jpg'})
```

Data management

We will se here important stuff about data management in MATLAB without the use on any toolboxes like managing variables workspaces, importing/exporting *.csv or *.xls, *.xlsx files.

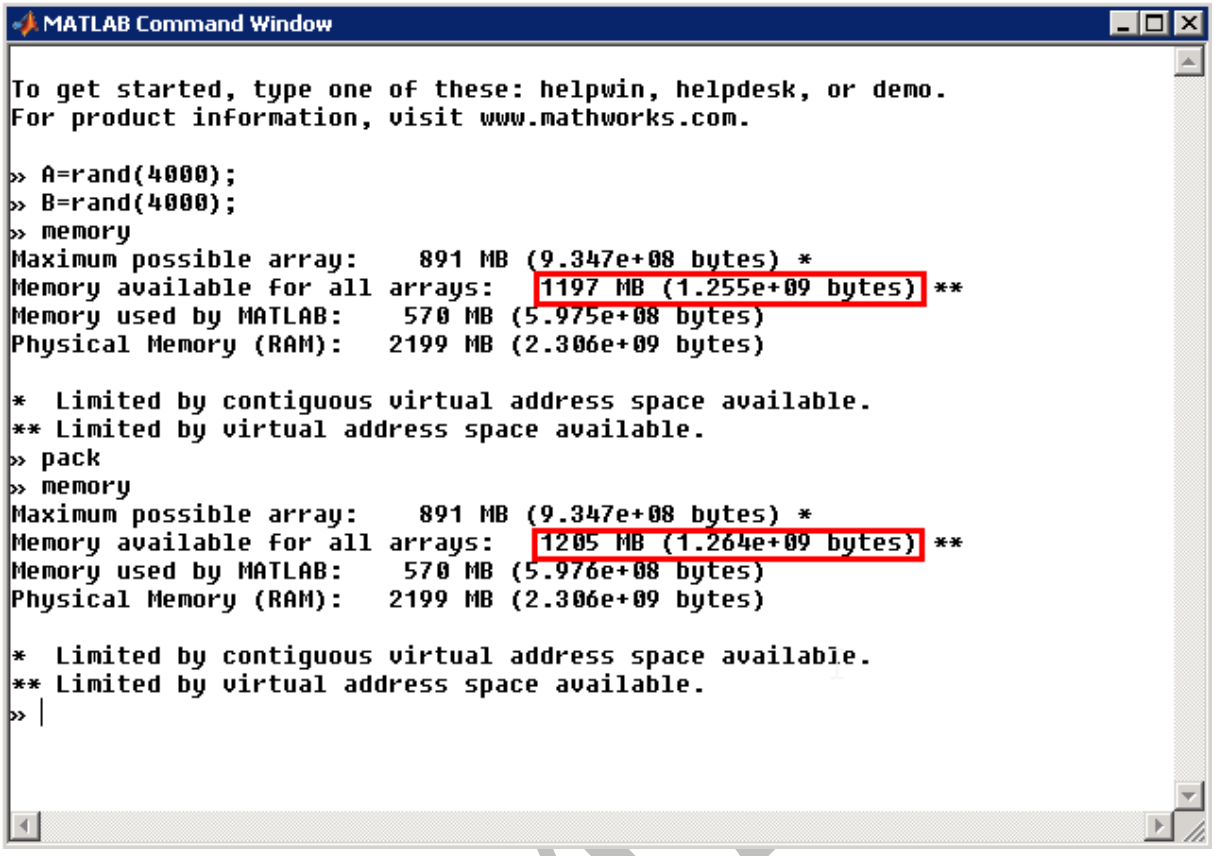
MATLAB files (*.mat and *.m)

This part is more a refresh than anything else on how to save a current variable workspaces and reload it again:

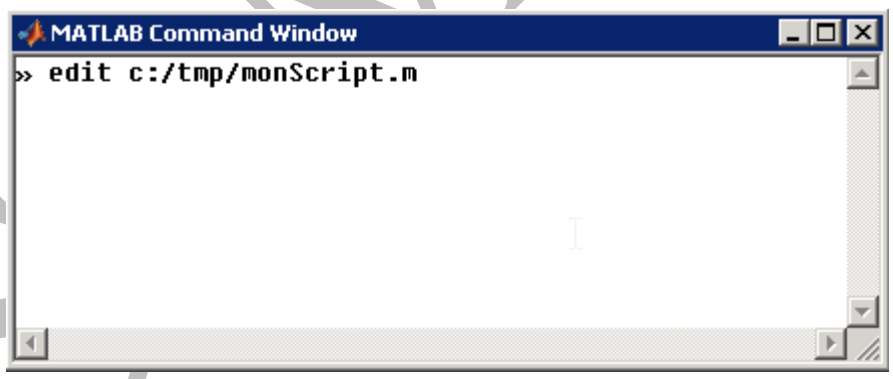


SA

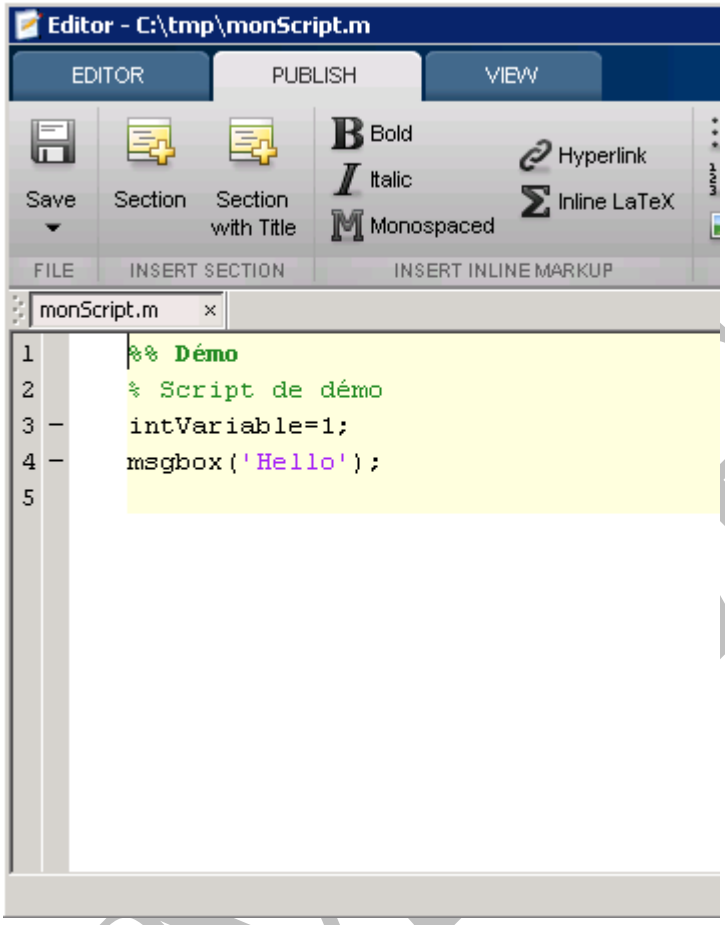
And also how to take a lock to the memory usage and defragment it:



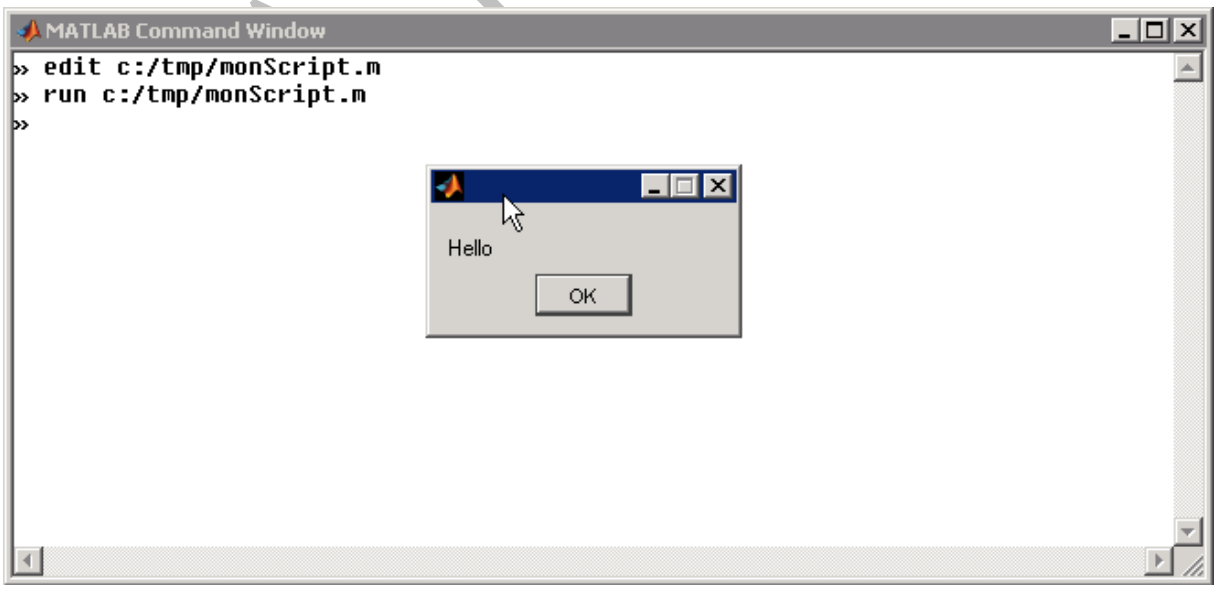
If we create a script:



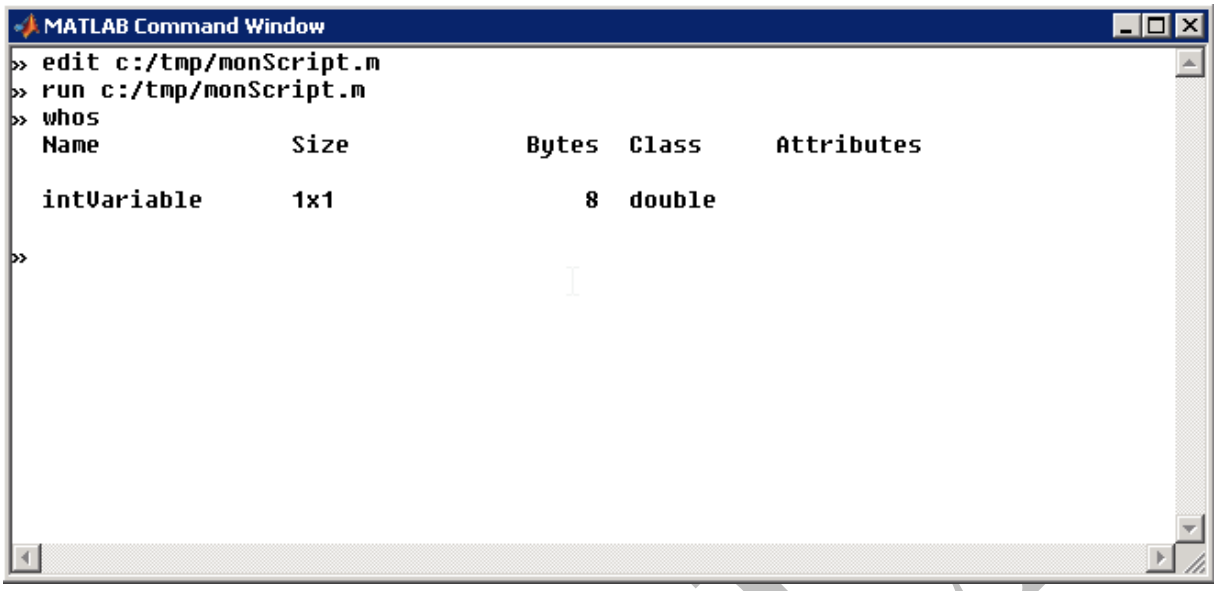
and that we write the following very simple stuff:



And that we run it afterwards:



Then we can see that the variable *intVariable* is now in the workspace:

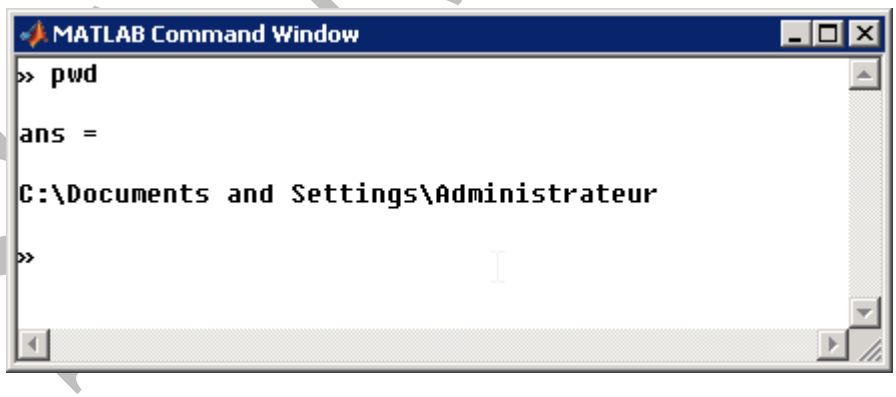


This is almost logic and not surprising that a variable into a script that is not a function can also be accessible from the current workspace.

Manage working paths

When you run a script or a function by default MATLAB will look into all defined working paths.

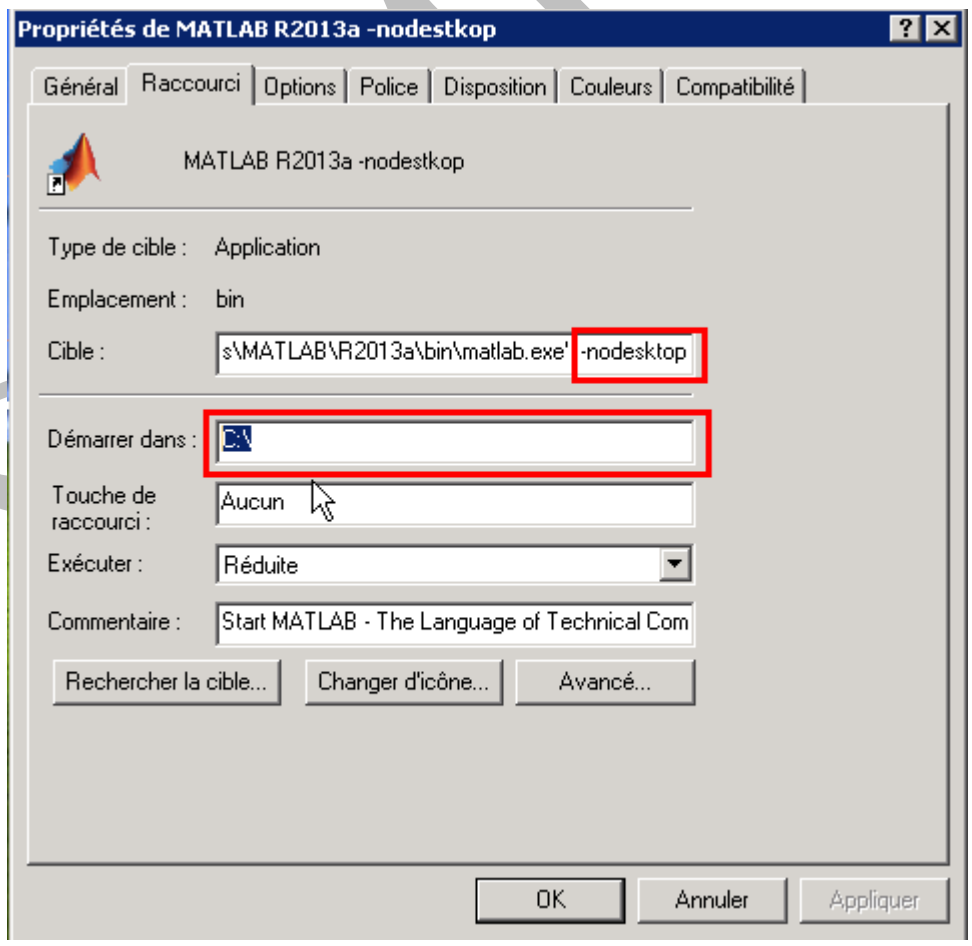
For refresh remember that to see the actual main working path you have to use the command **pwd**:



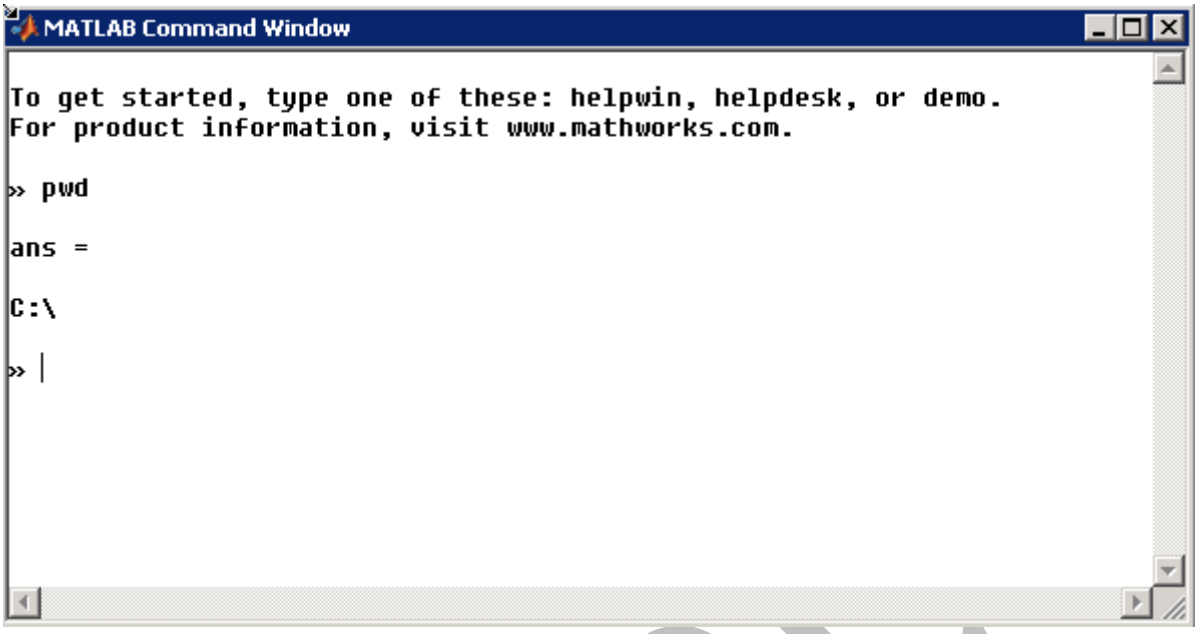
If you want to change this default path that is used especially for the *startup.m* and *finish.m* script files you how to customize the MATLAB shortcut as shown on the next page:



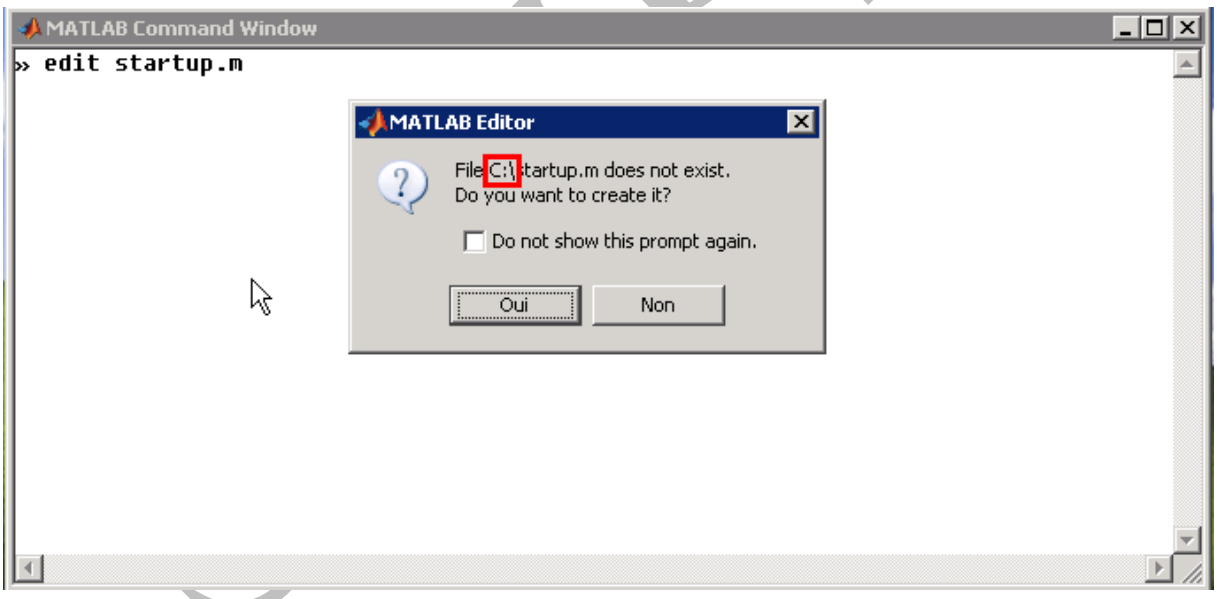
Make a right click on the shortcut and go into its properties and after in the shortcut register change the property *Start in* (the *–nodesktop* is facultative):



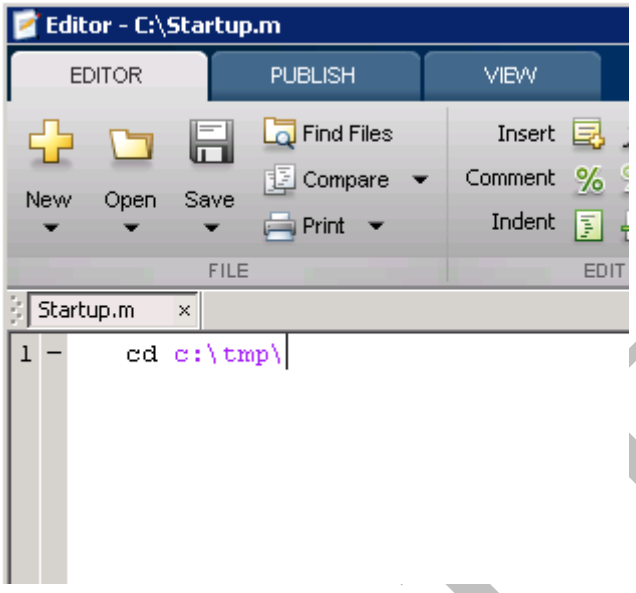
Then you will see that the current main working path has changed:



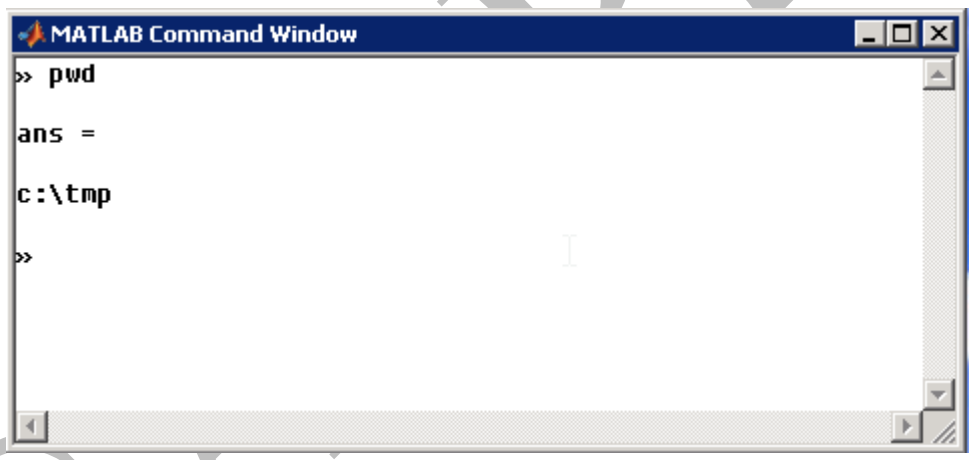
Now that the default working path is change you can create or move your *startup.m* file into it:



And write whatever you want:

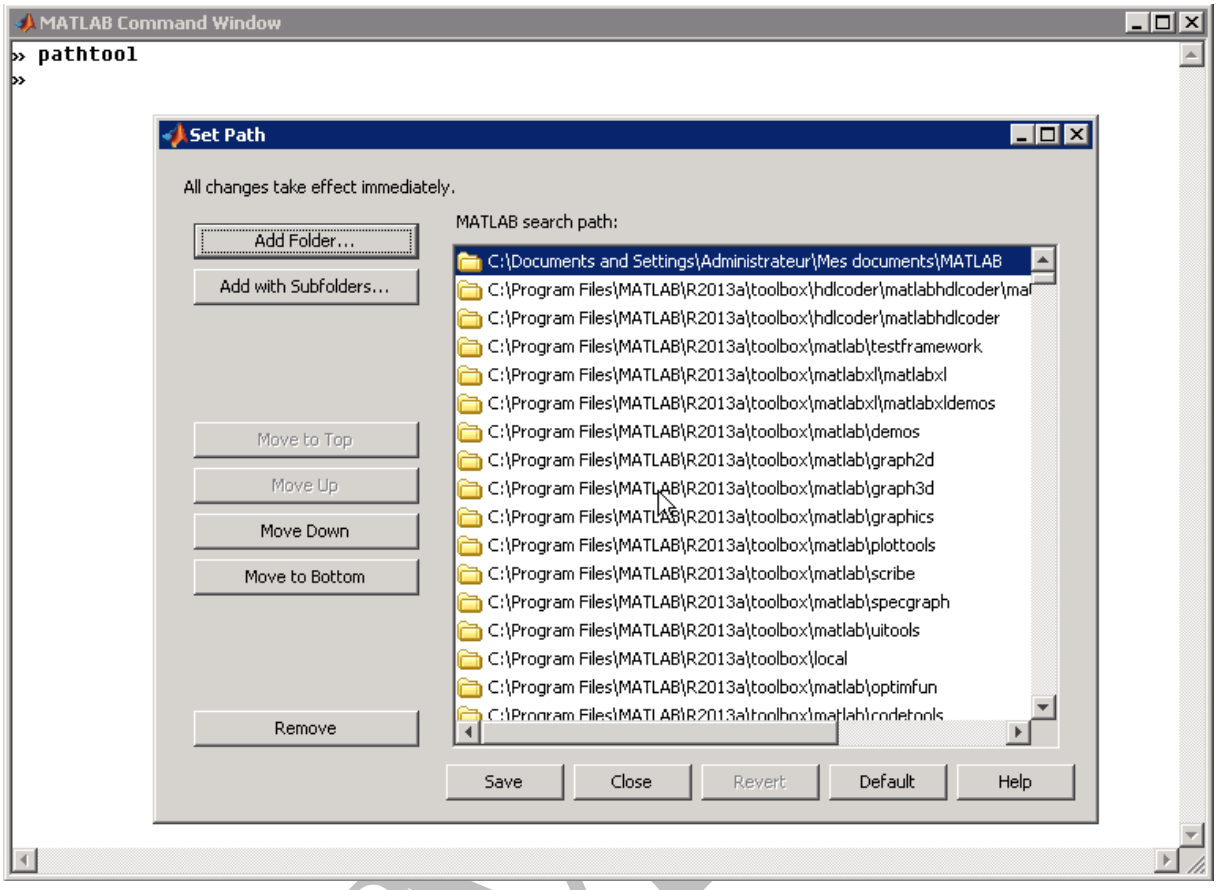


and then if you close and restart MATLAB you will see that the *startup.m* file will be executed automatically from its new location:

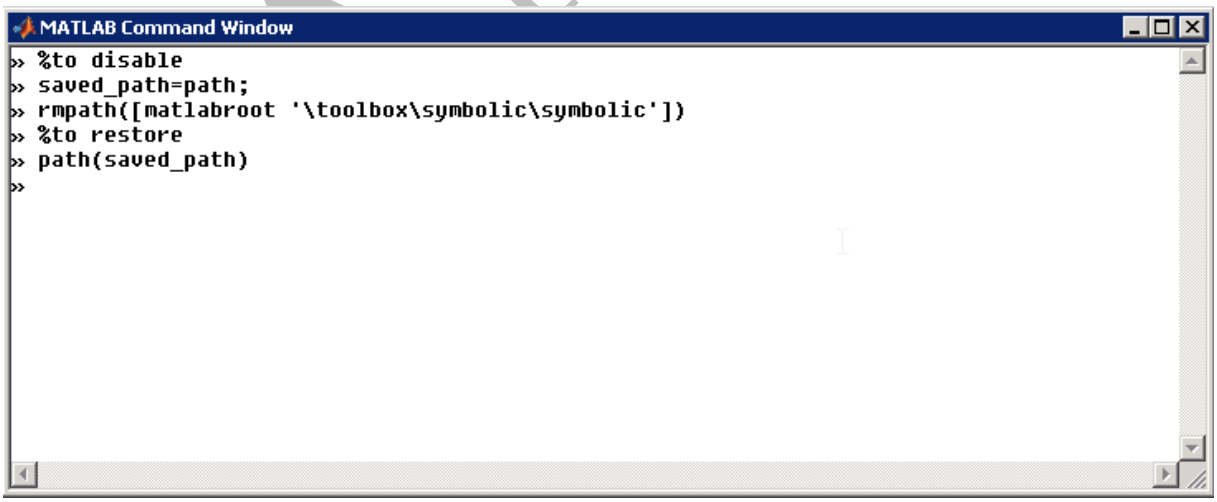


SAMPLE

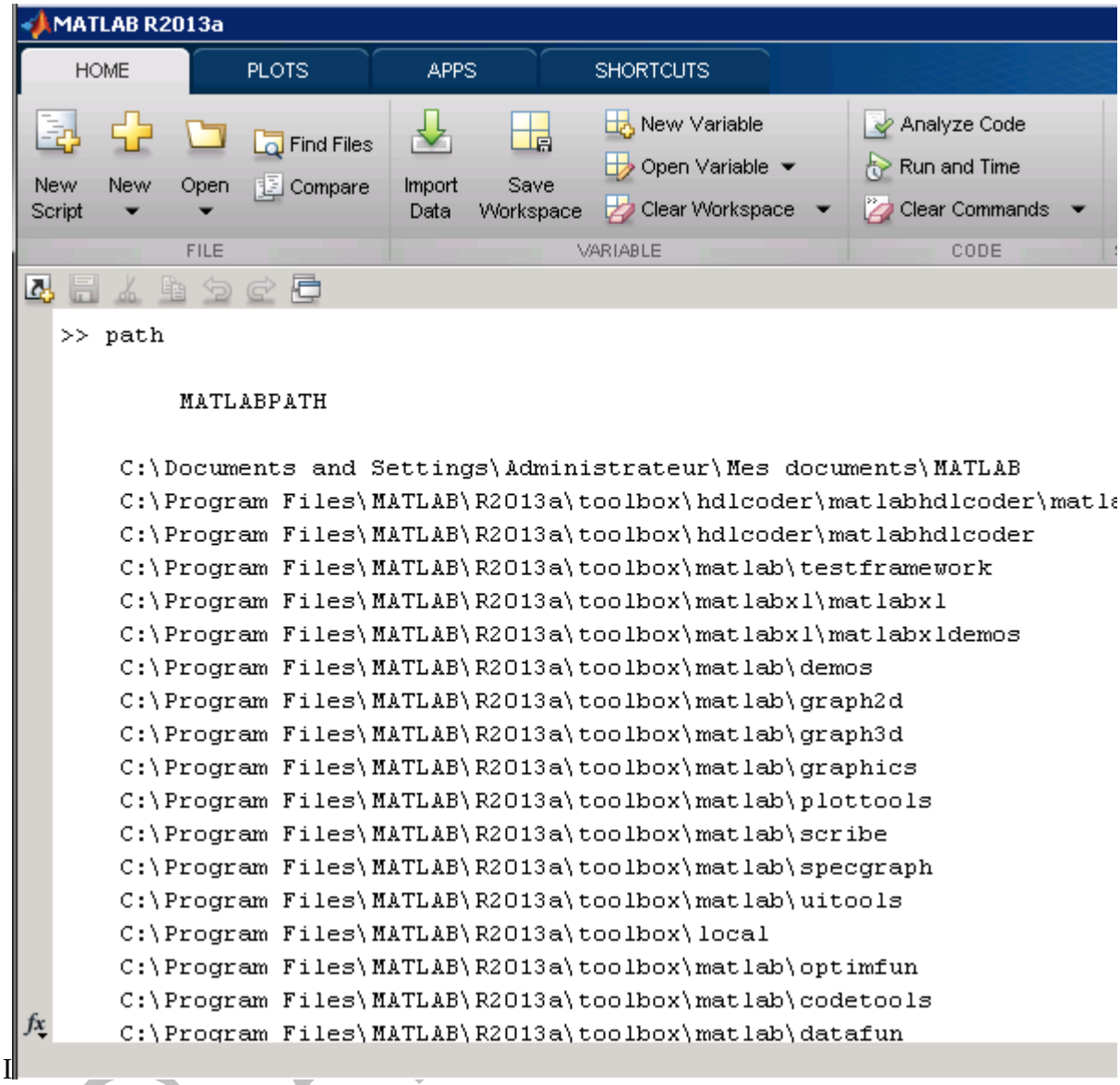
You can also manage working paths with a convenient user-friendly tool called the "path tool" by typing **pathtool** in the *Command Window*:



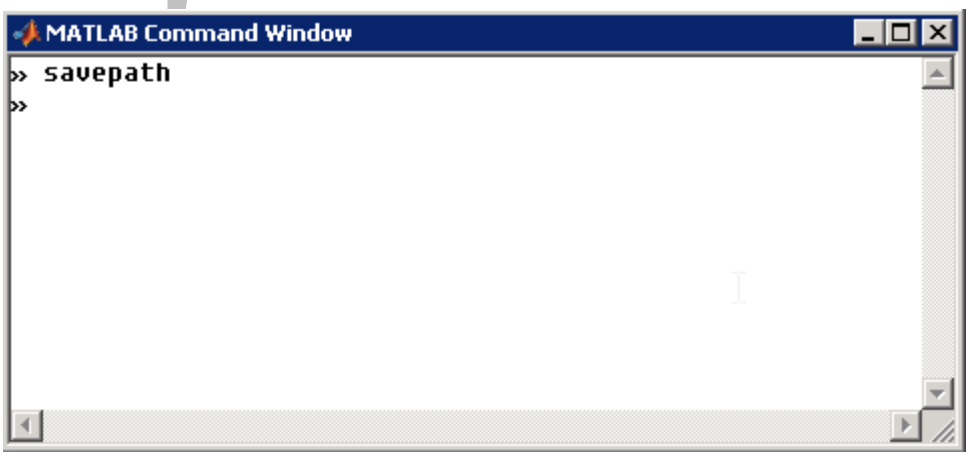
It is very easy to use thus there is nothing special to say. If you prefer to remove a Toolbox by a command you can still use:



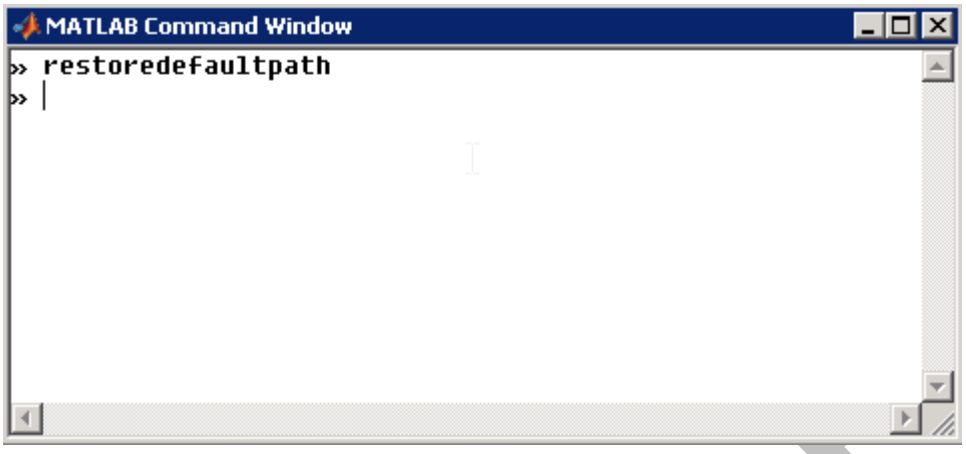
If you type the command **path** you will see the list of all active paths in you MATLAB:



and if you want to save the status of your news paths for a future session of MATLAB you will have to not forget to type the command **savepath**:

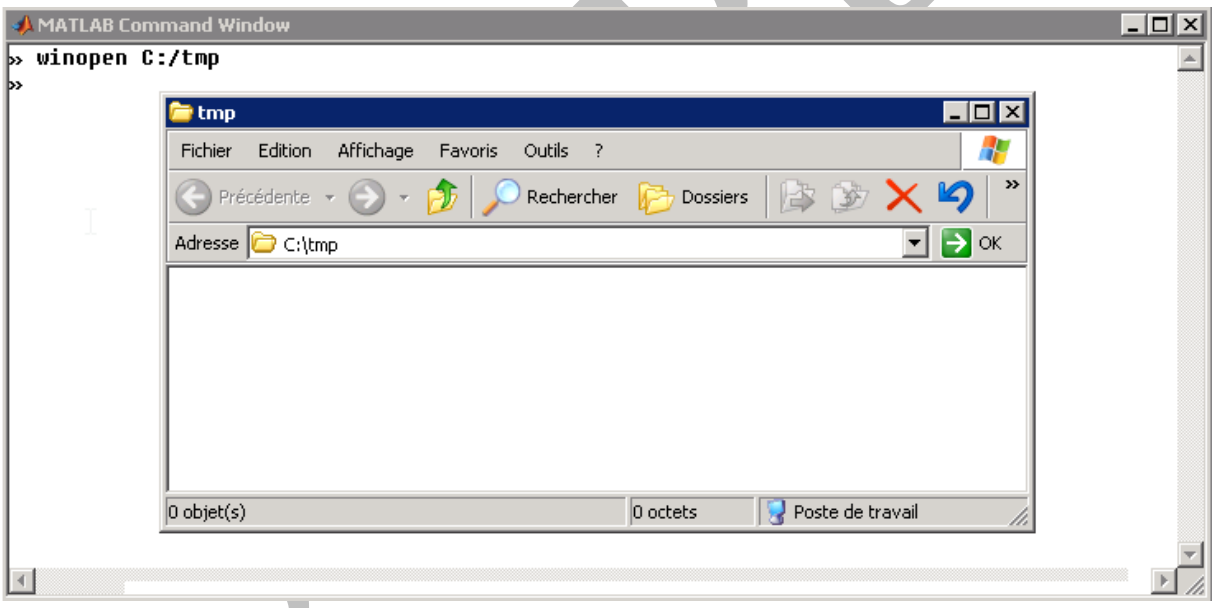


At any moment you can restore MATLAB defautl factory working paths if necessary by typing the command **restoredefaultpath**:

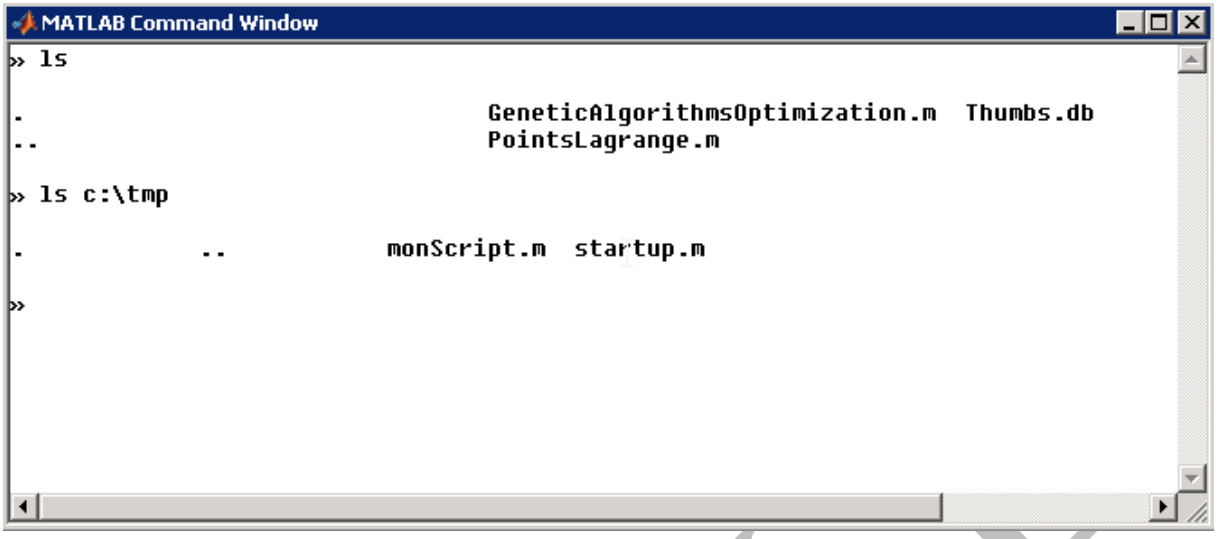


Browse and manage folders and files

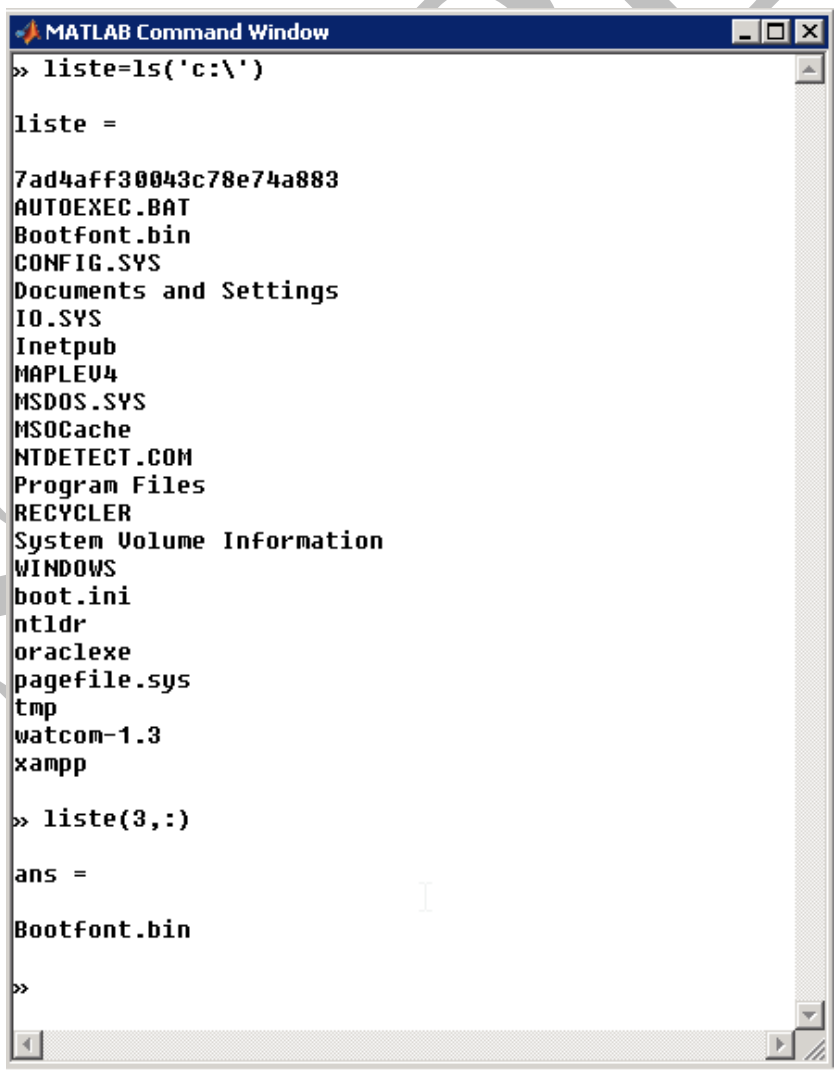
When you create scripts that autogenerate reports it can be useful to open for the user the folder where all reports were saved. You can easily do that thanks to the command **winopen**:



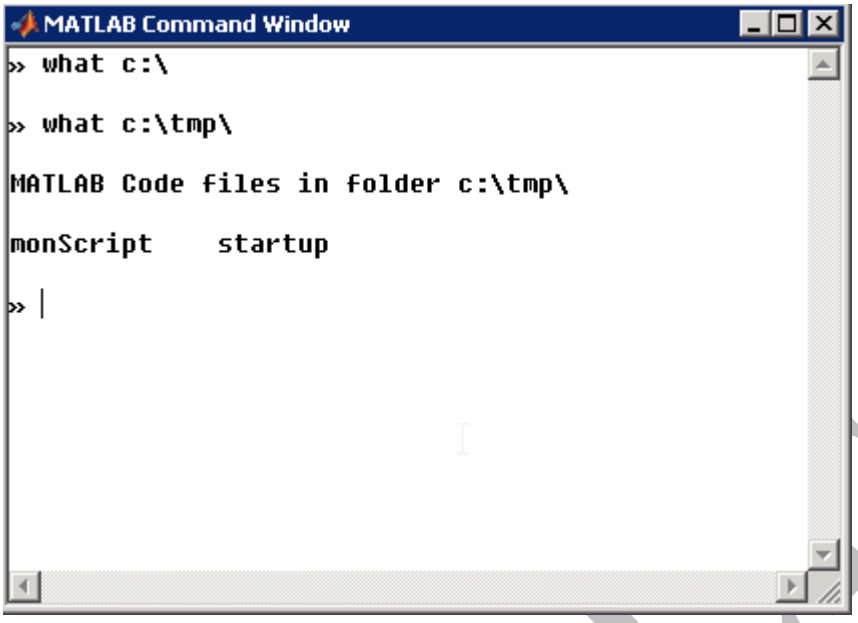
To list all the files in a given folder just use the command **ls** like in UNIX:



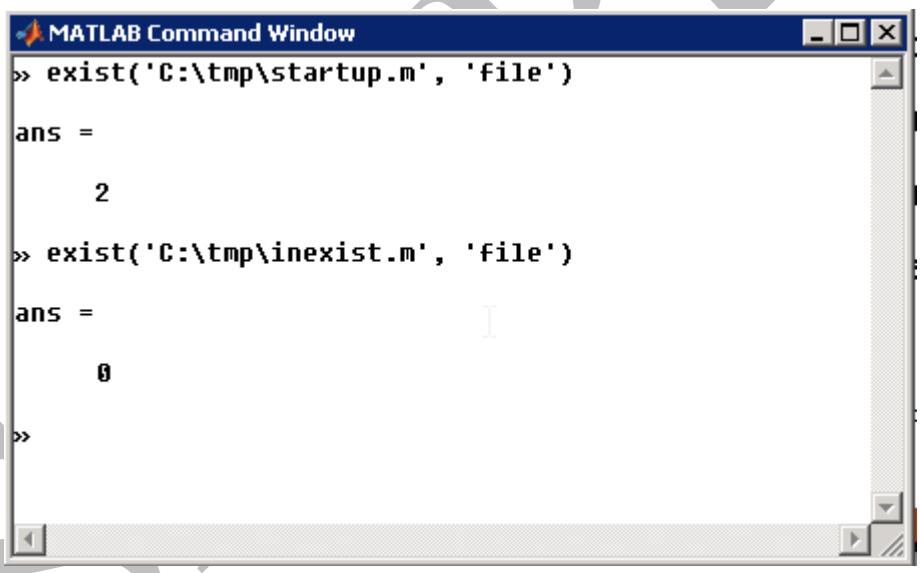
You can also store the list of files and folders in a vector to use it later for treatment purposes in your future scripts:



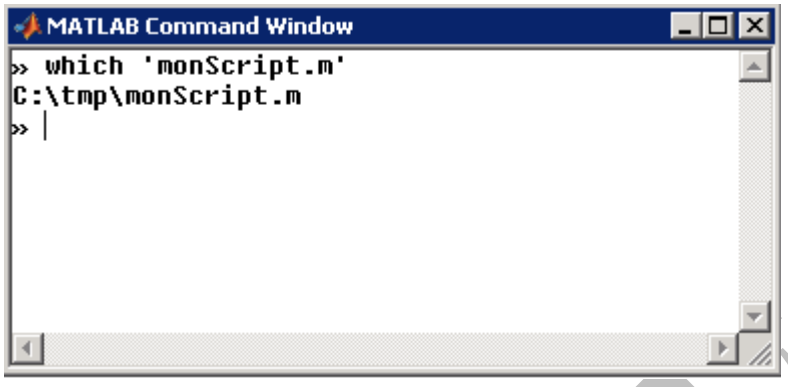
If you use the command **what** instead of **ls** you will get only MATLAB files type:



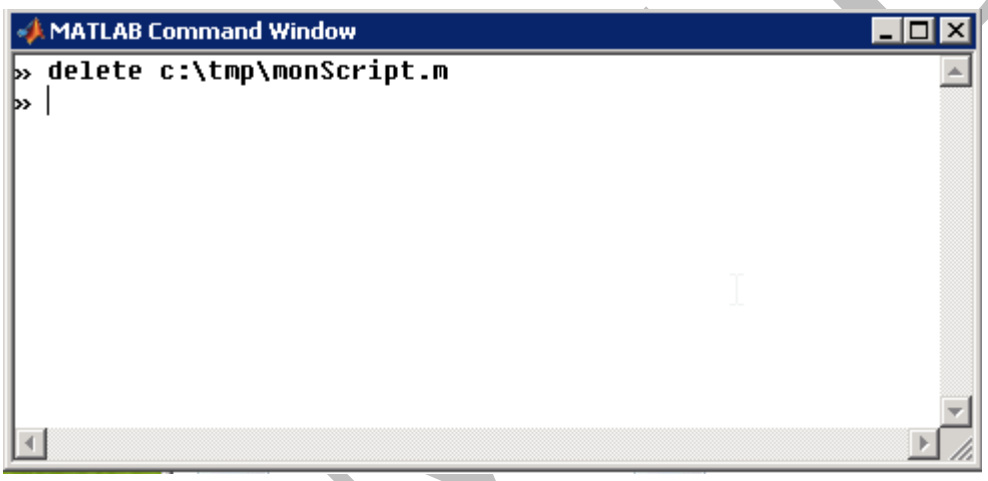
You can also check if a file already exists or not:



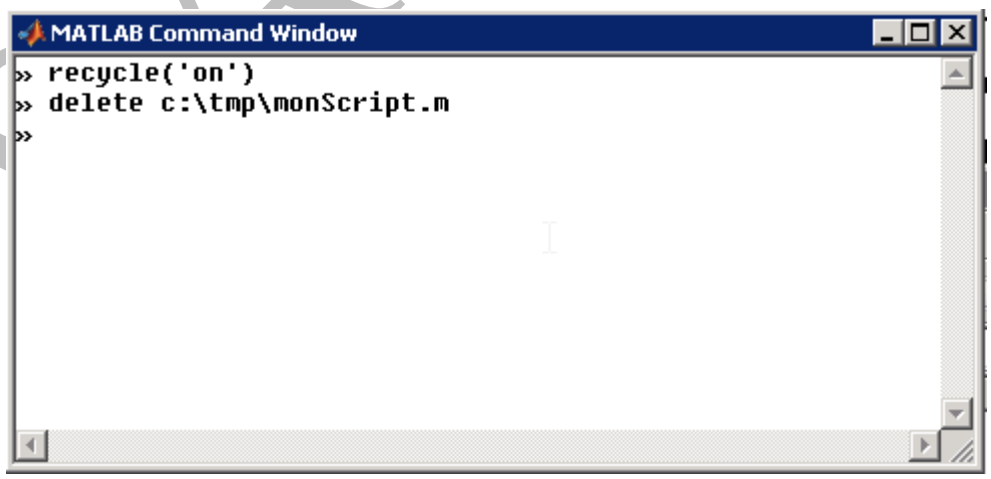
If you use the command **which** with a file name **or** a function name this command will give you in which working path is stored the corresponding script:



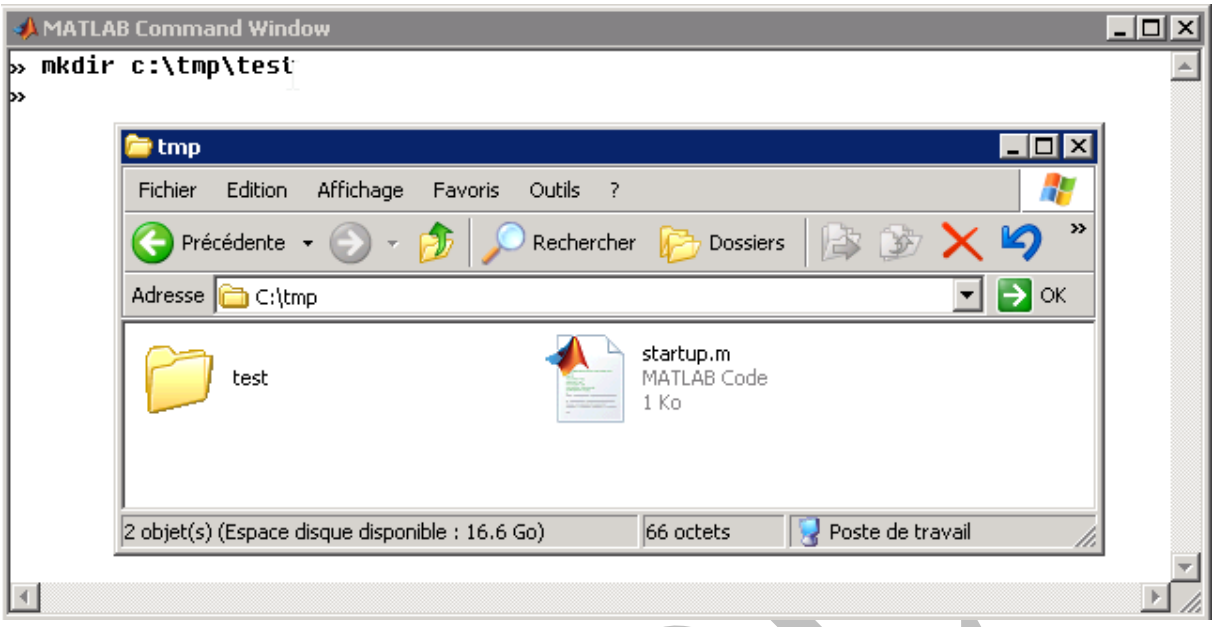
You can also delete a file using the command **delete**:



The problem is that the file will be definitively deleted. If you want to activate the recycle bin you have to activate it by using the command **recycle('on')**:

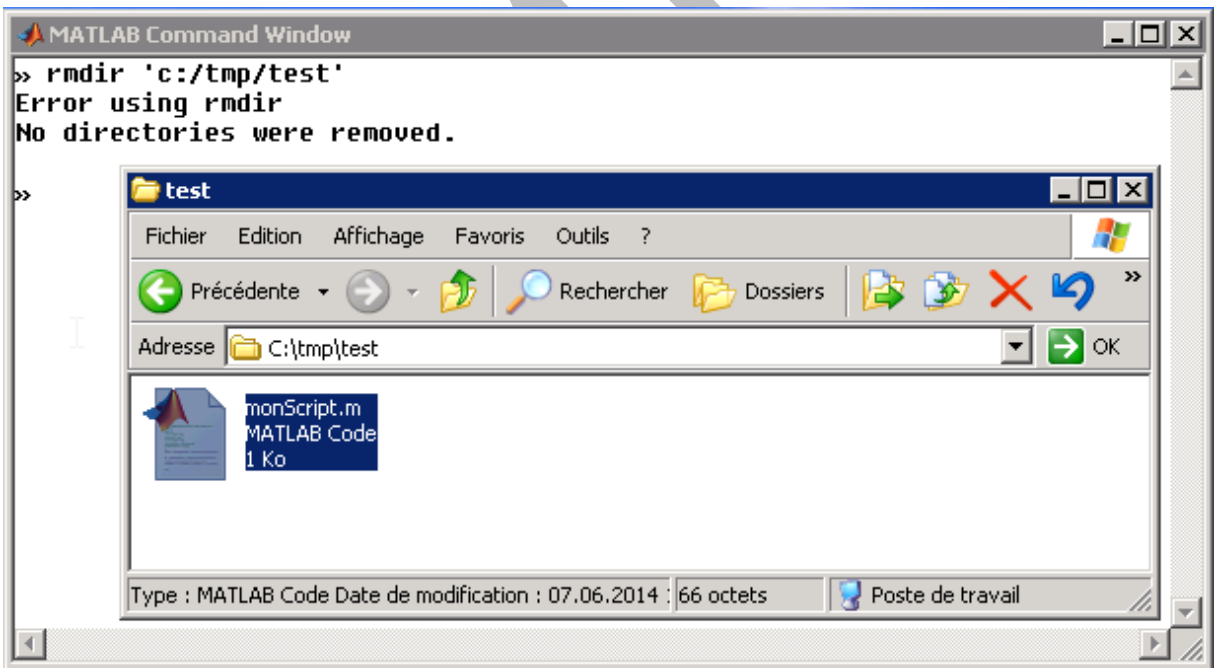


You can also create folders by using the UNIX **mkdir** command:

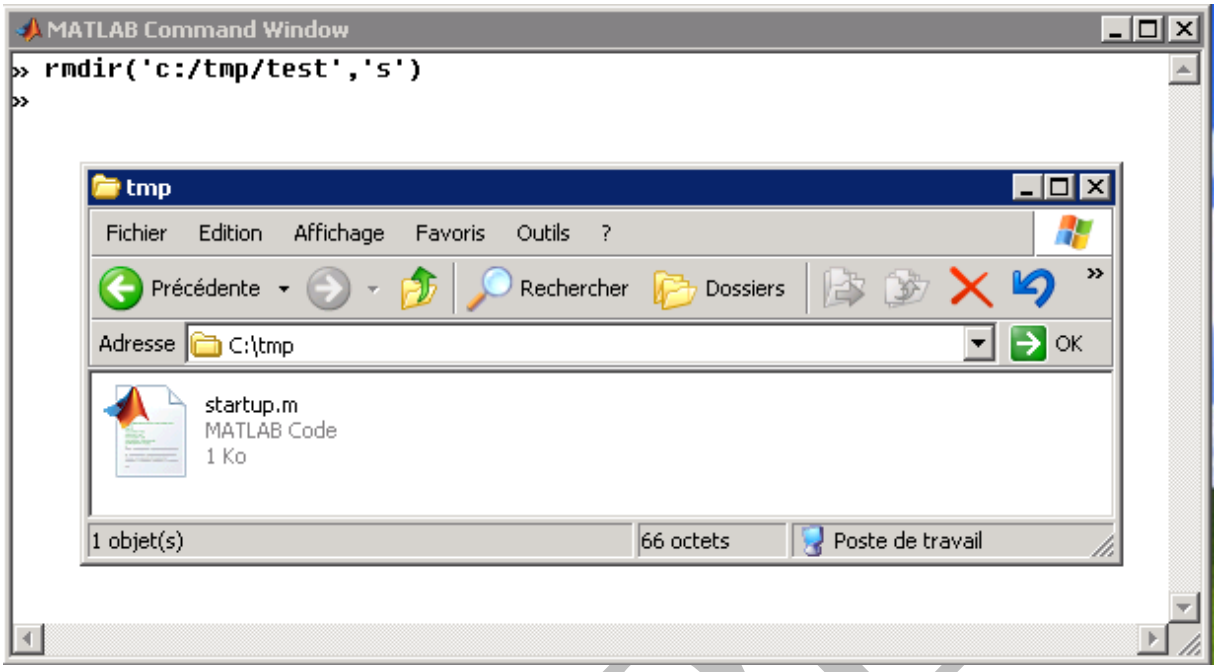


Or remove (delete) a folder by using the UNIX **rmdir** command.

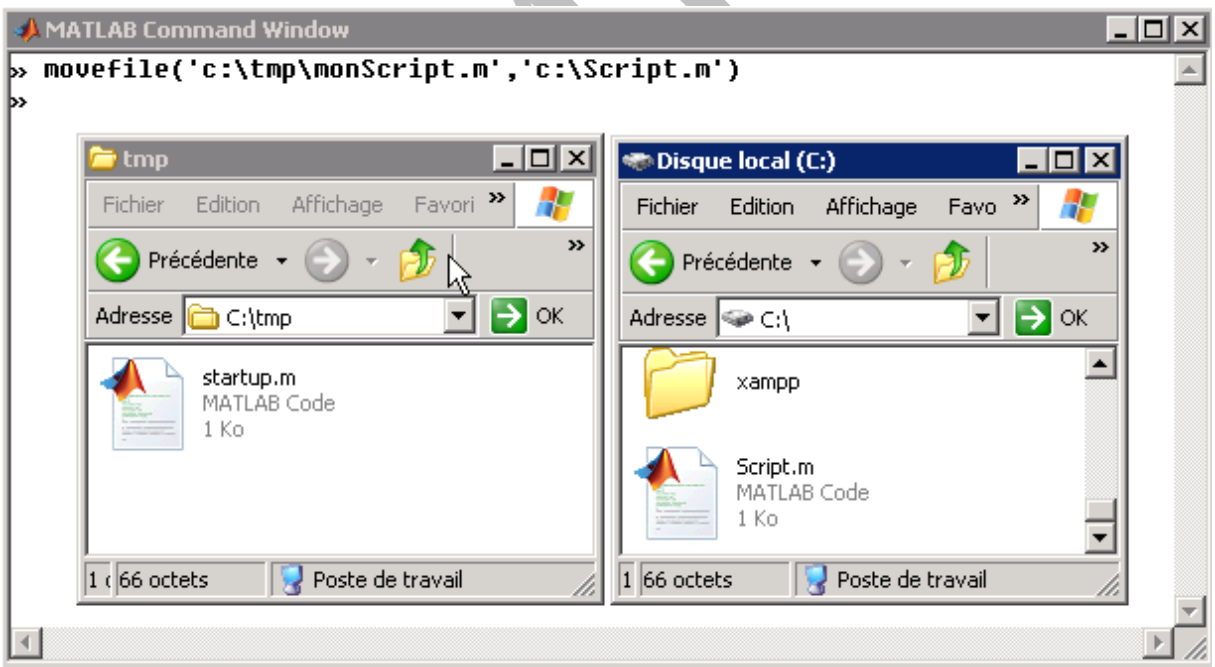
But there is a small trap. If you try to simple use the command you will get an **error message**:



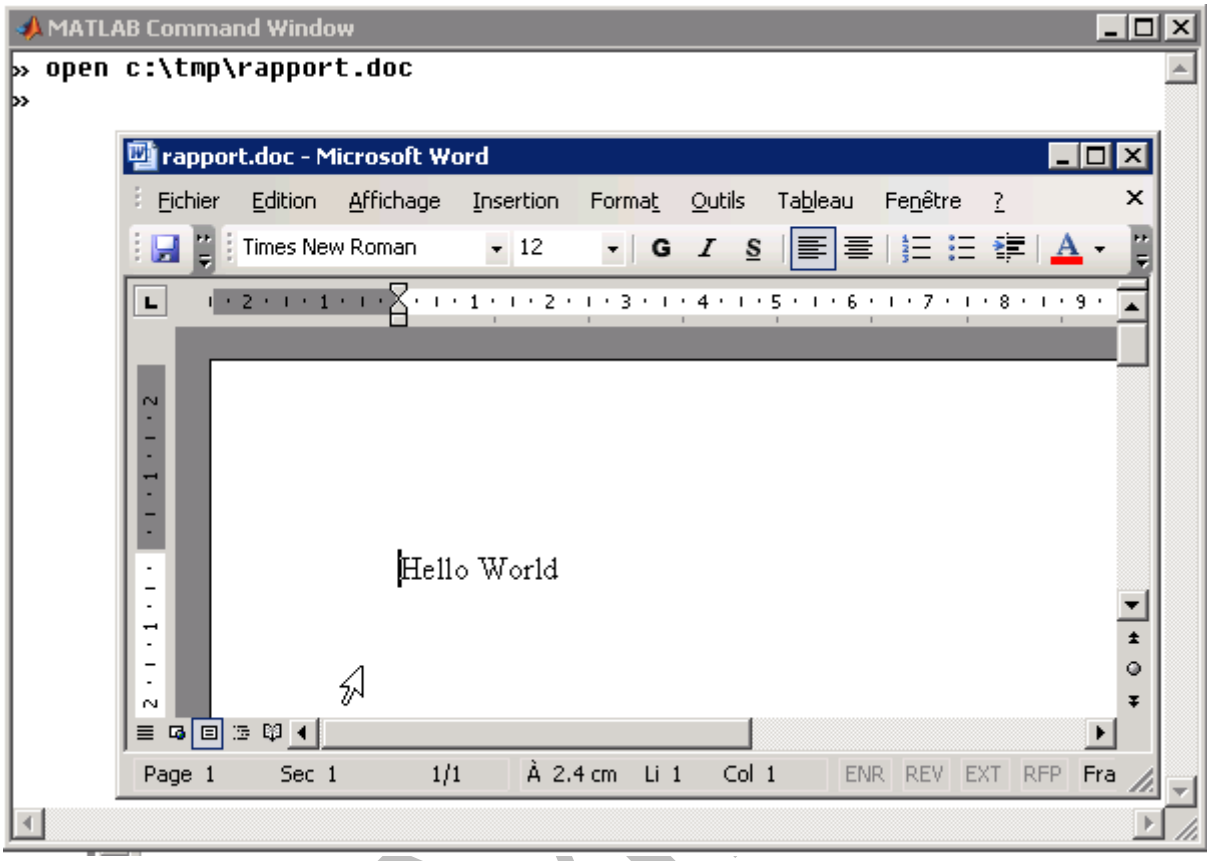
To avoid this you have to pass the parameter 's' to the command like this:



You can also need when you develop applications with MATLAB to move files from a folder to the other one. For this purpose you can just use the function **movefile( )**:



And if instead of opening a folder with reports you want to open the reports themself just use the command **open**:



SAMPLE

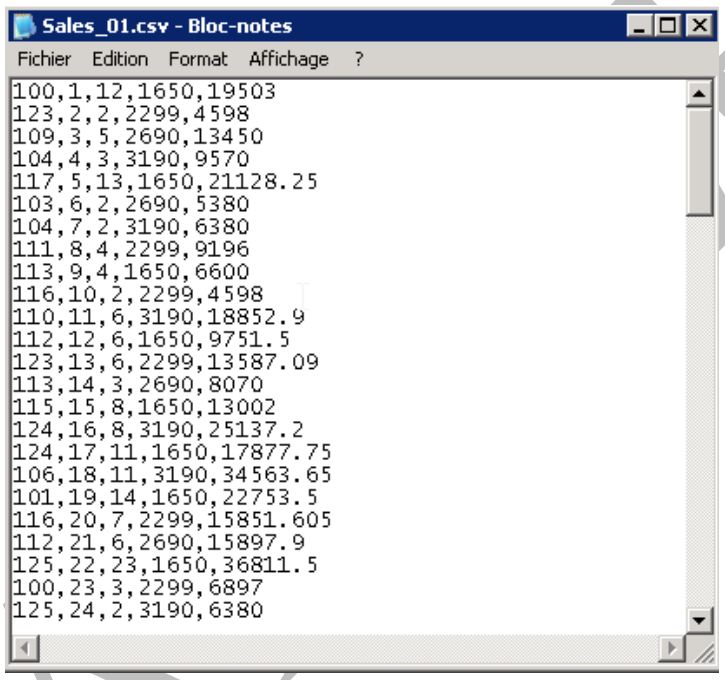
Import/Export csv files

We will see here how to import *.csv files without using the import wizard of MATLAB but just by using the *Command Window*.

Import very clean local csv files (with comas, without headers, without texts, without dates)

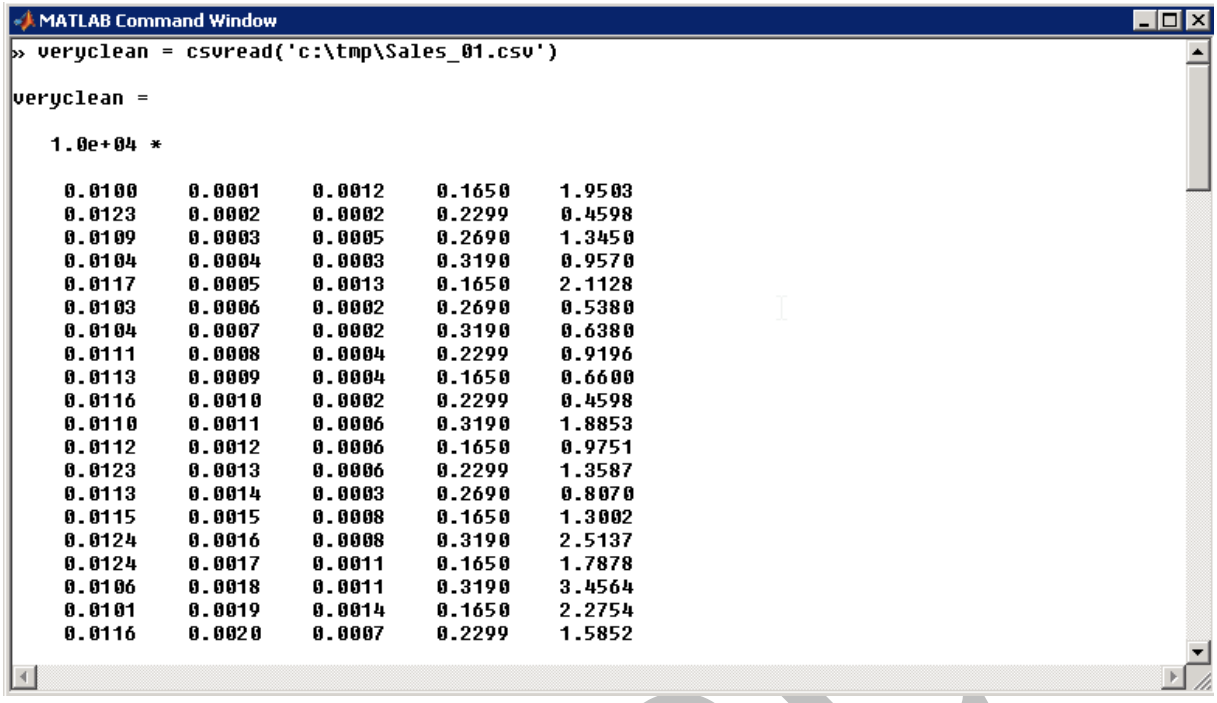
The command **csvread( )** can't import a file with semi-colons or with headers or with texts or with dates.

You must have a file with comas and numerical values only:

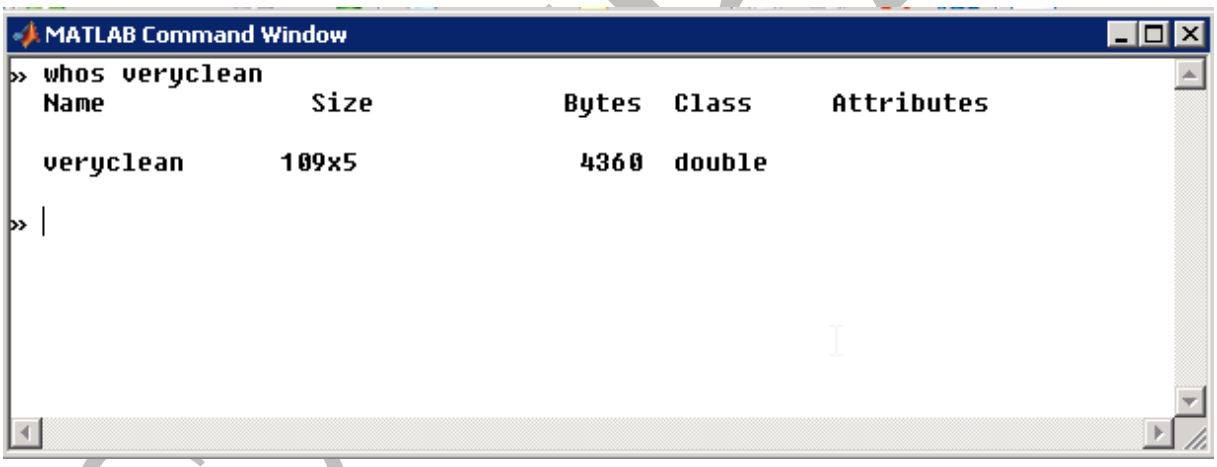


Then you just have to use the command **csvread( )** to import the datas into a matrix:

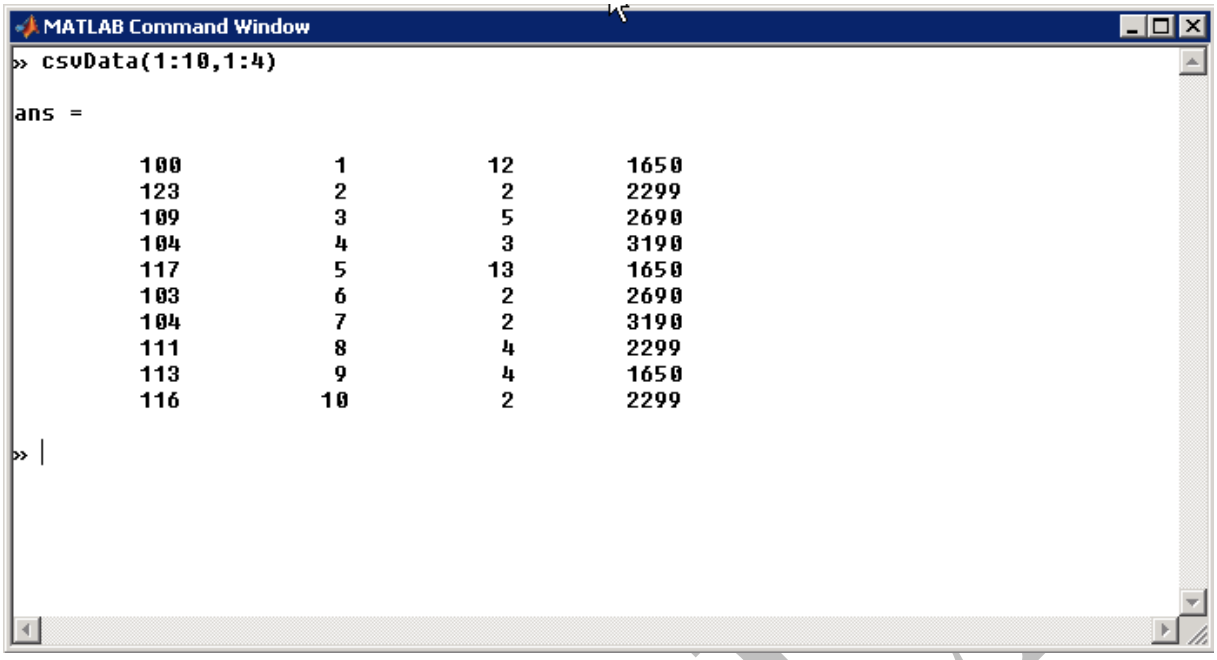
SAMPLE



As you can see it's a matrix:



And you can like any other matrices take a subset:

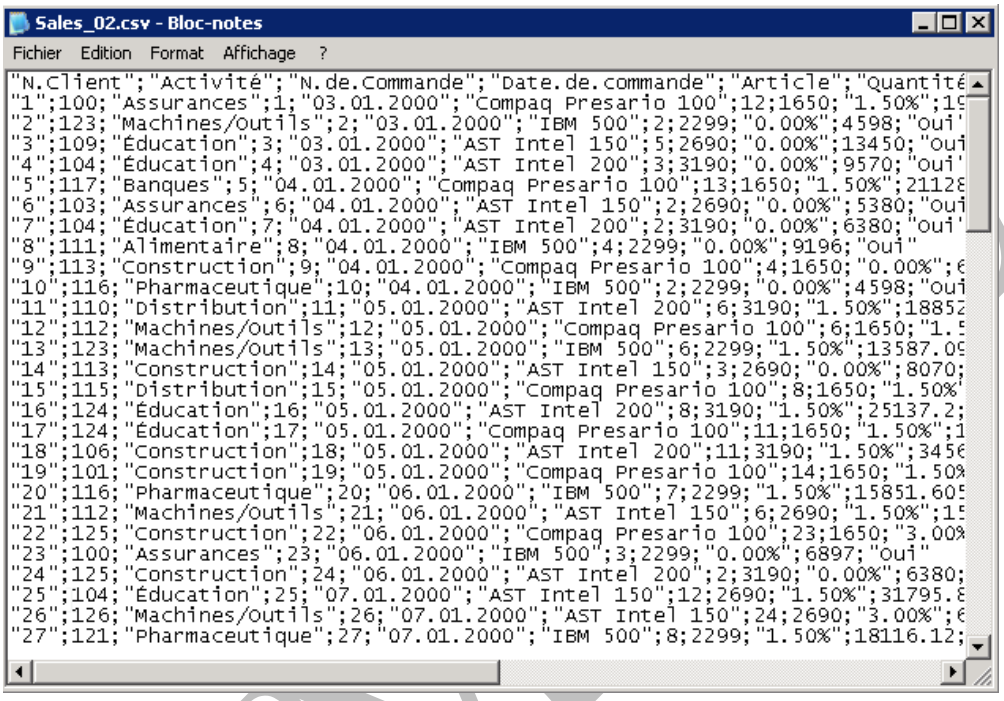

MATLAB Command Window
» csvData(1:10,1:4)
ans =
100 1 12 1650
123 2 2 2299
109 3 5 2690
104 4 3 3190
117 5 13 1650
103 6 2 2690
104 7 2 3190
111 8 4 2299
113 9 4 1650
116 10 2 2299
»

SAMPLE

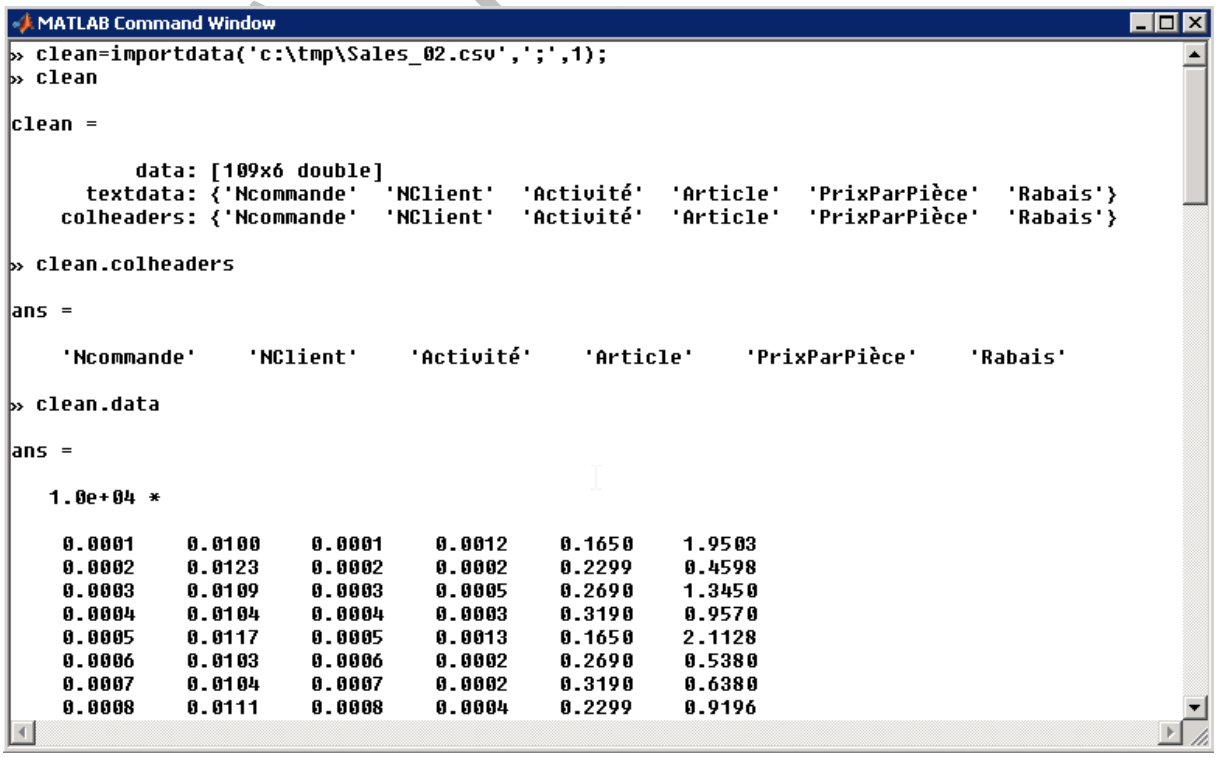
Import clean local csv files (with semi-colons, with headers, without text, without dates)

Now we will see that the function **importadata( )** can manage a few more things than **csvread( )** but still can't import files with strings or dates.

Consider the following file:

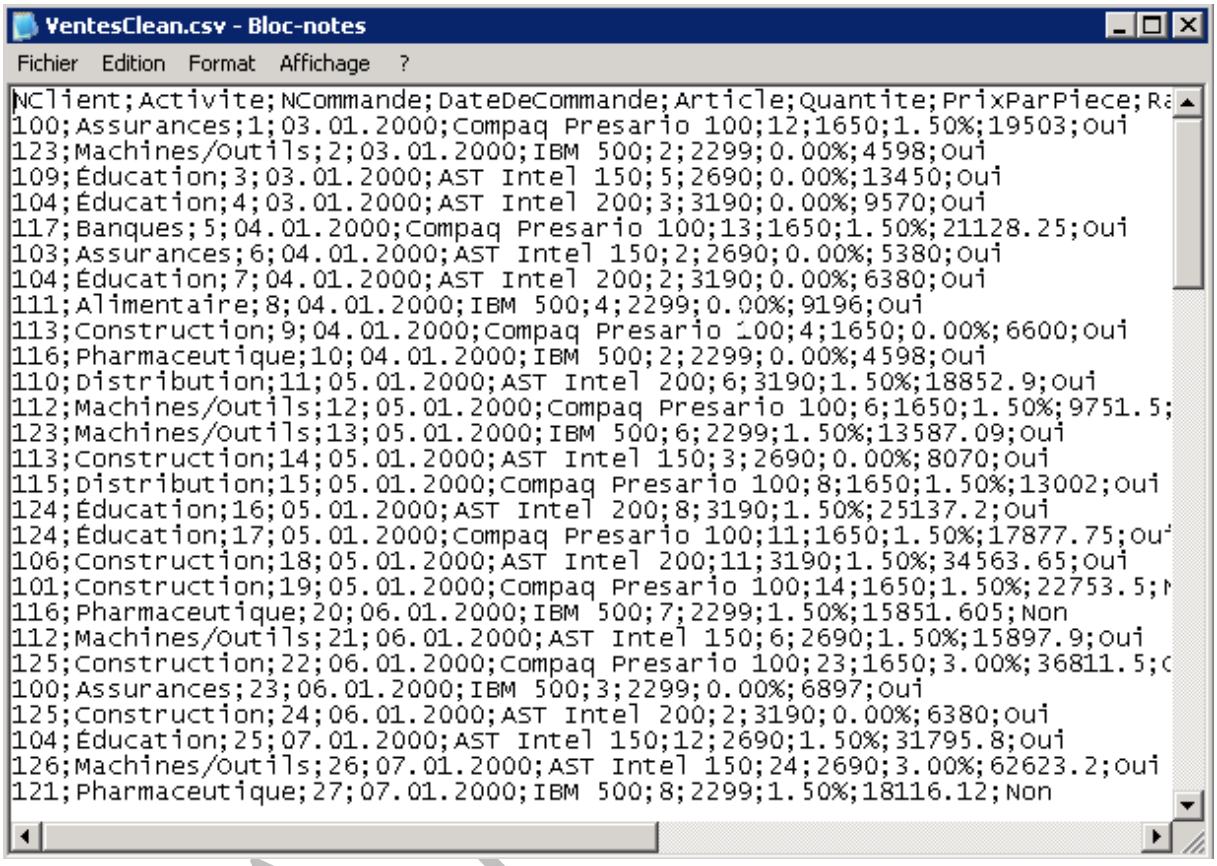


The result will be easy to manage because it results in a structure:



Import local csv files (with semi-colons, with headers, with text, with dates)

The most easy way to import typical csv files is to use the function **fopen( )** associated with **textscan( )**. Consider as example the following file:

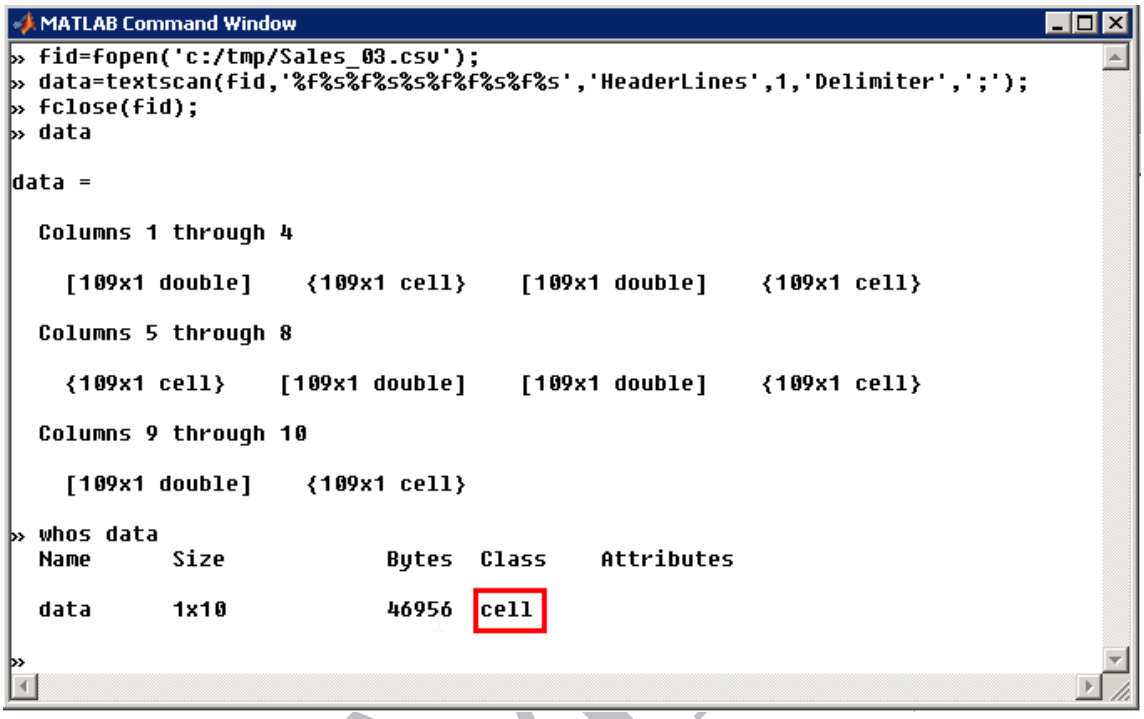


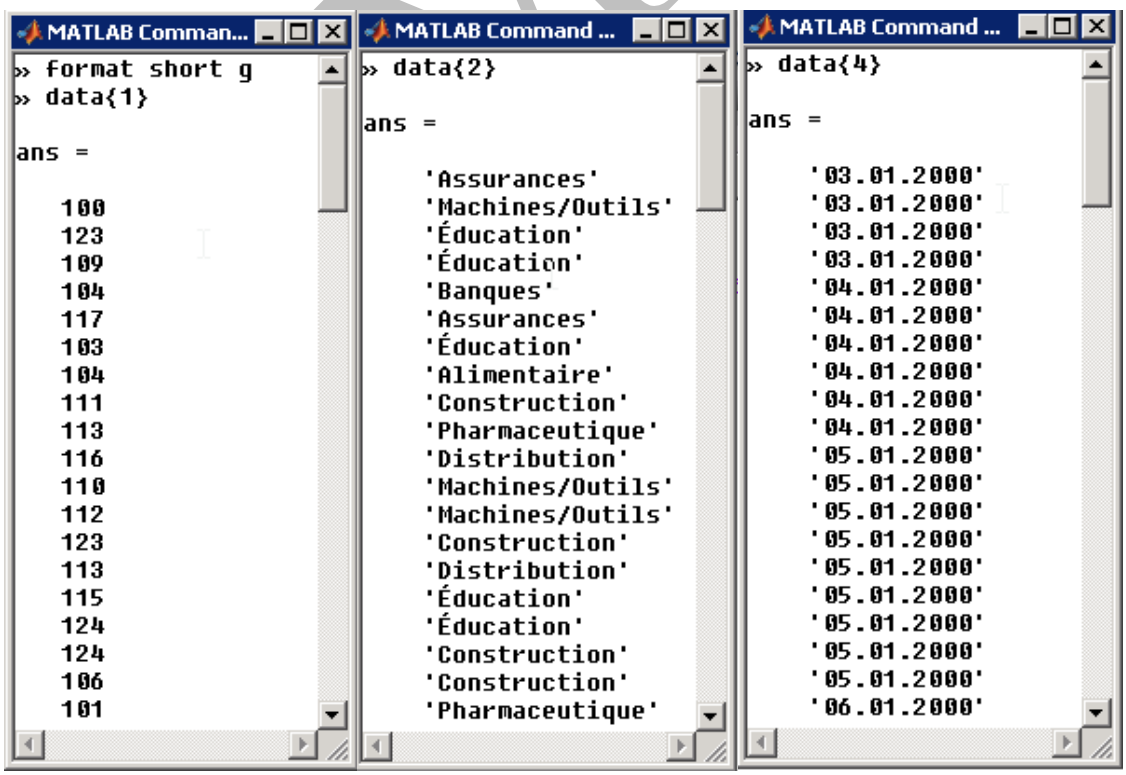
SA

Now the commands below tells **textscan( )** how is each row in your csv file to be converted to. **%f** means "64 bit double" and **%s** means "string" so for example:

'%f %f %f %s %s'

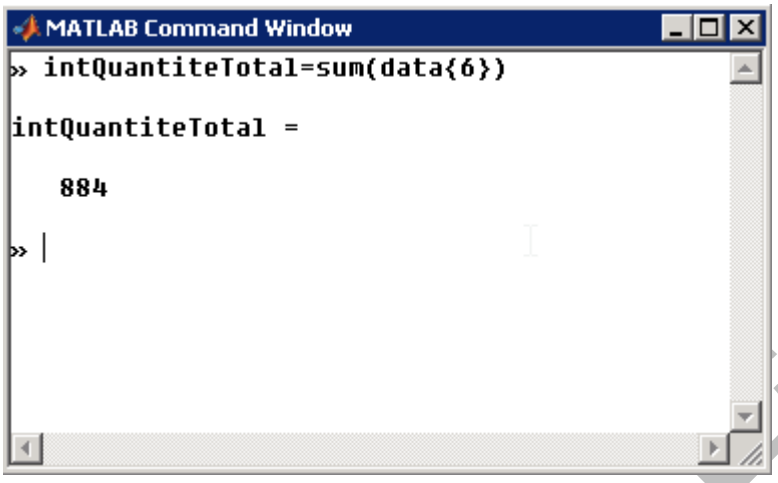
means "3 doubles followed by 2 strings". For tab separated files the delimiter is **char(9)**



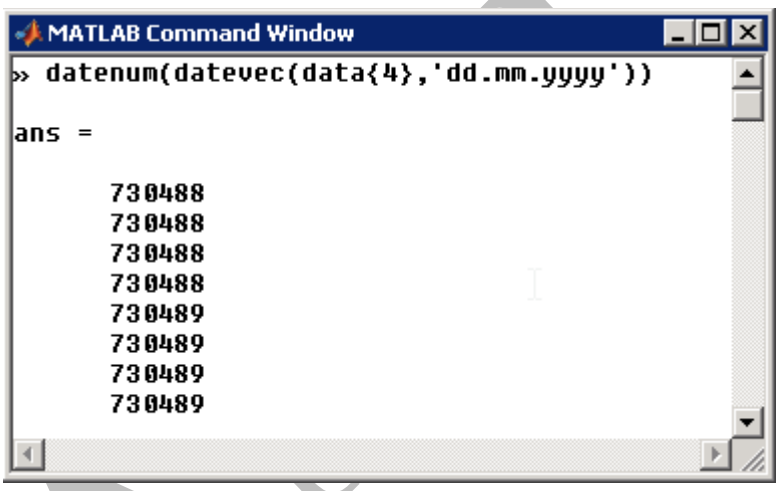


As you can see it the content is imported as a cell array!

We can style make calculations on numerical columns:



And convert dates strings into real dates using classical conversion technics:

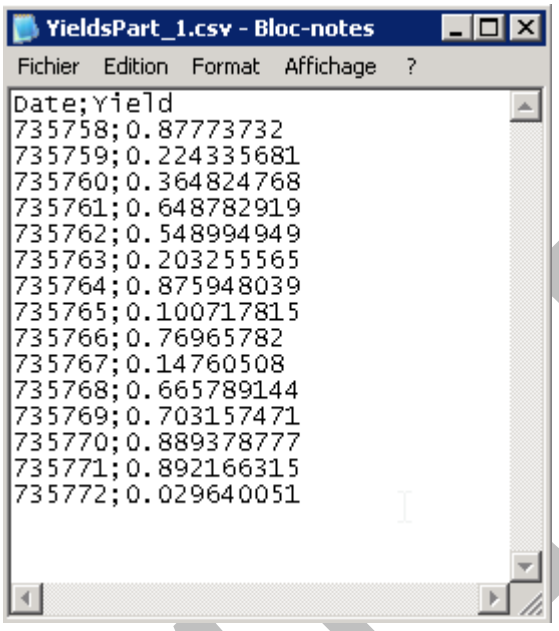


SAMPLE

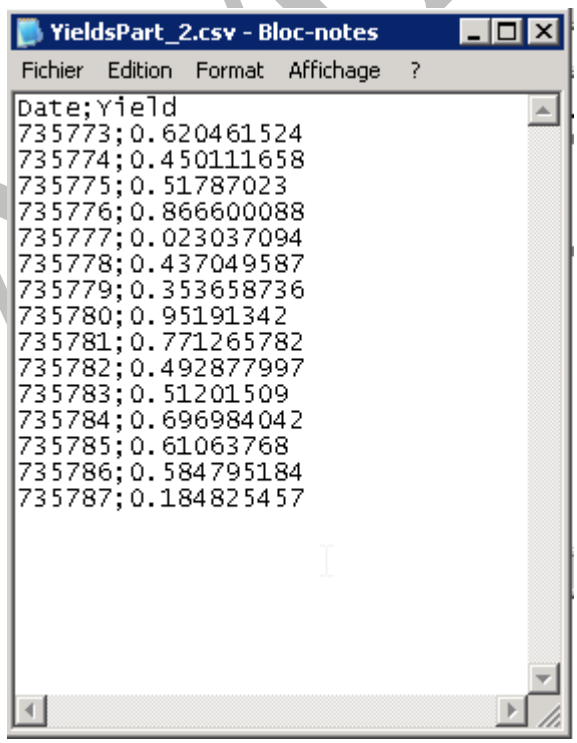
Merge csv files

You can normally always merge various files content.

Suppose we have the following file:

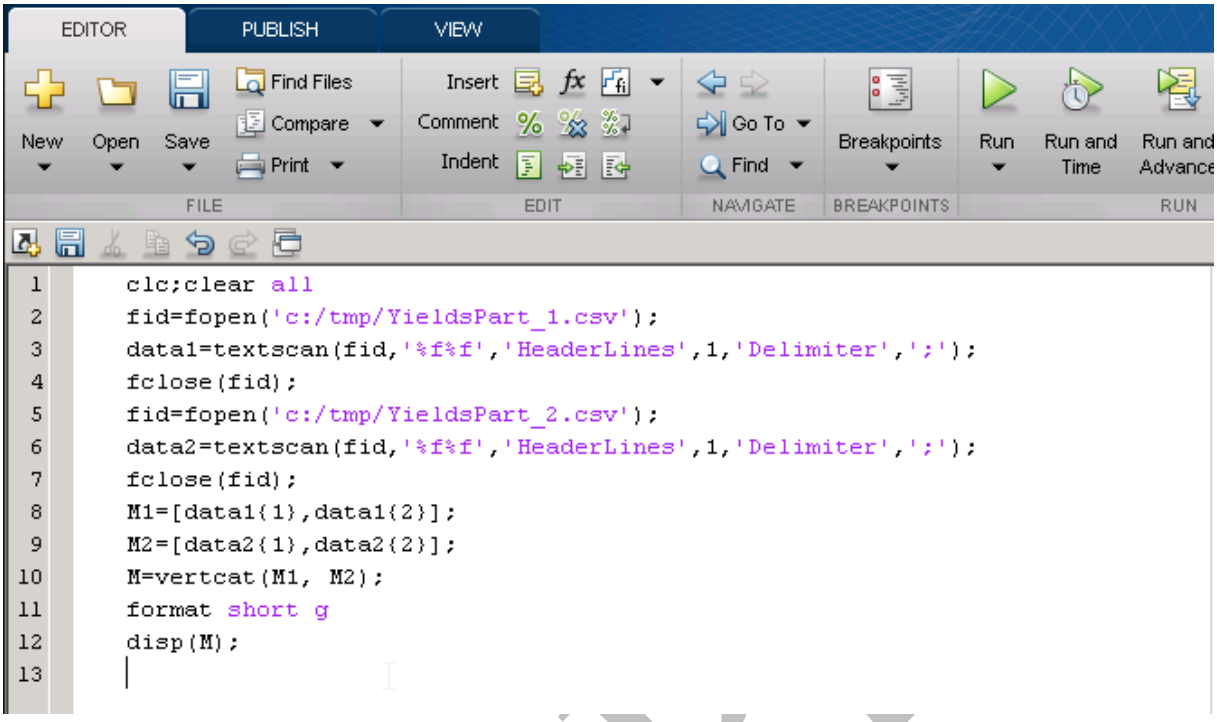


and another one:

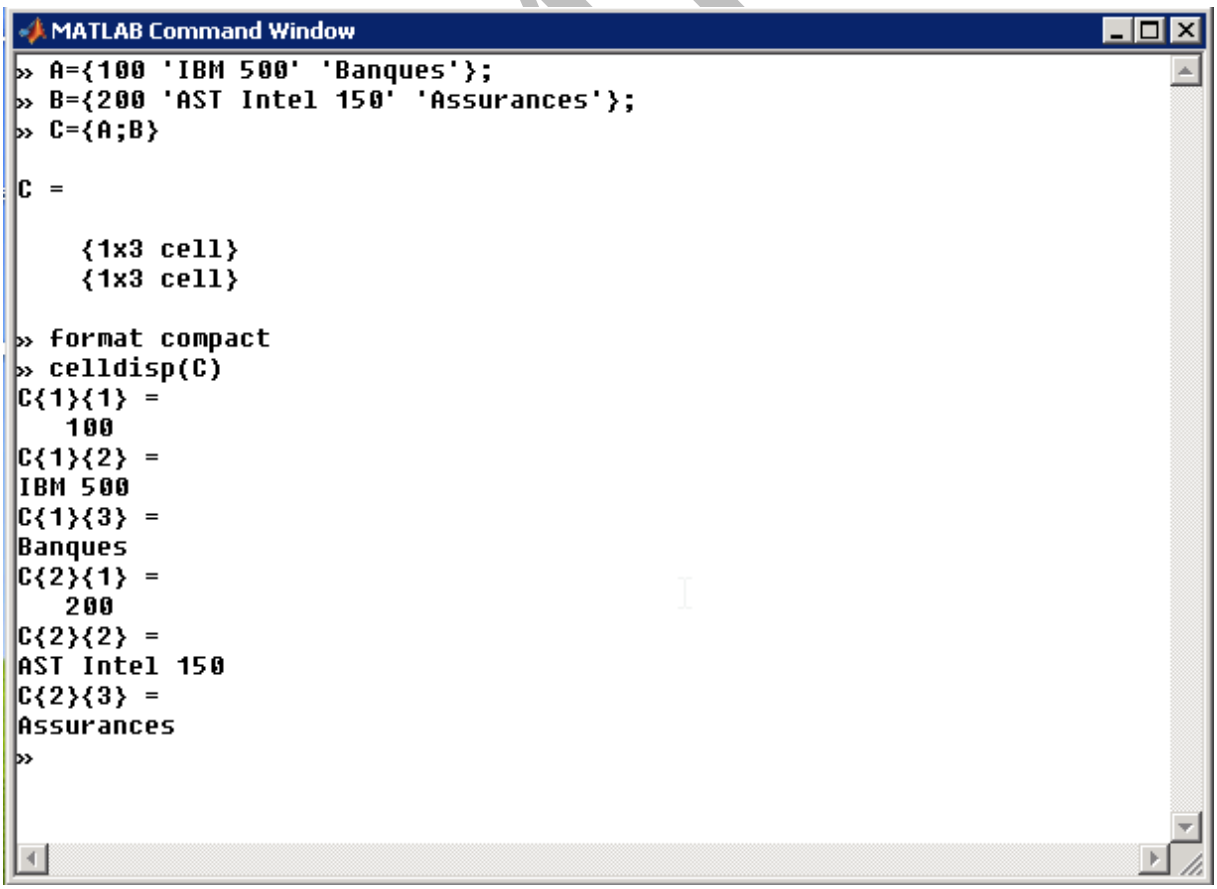


SAMPLE

We import the data and merge them together using classical matrices technics:

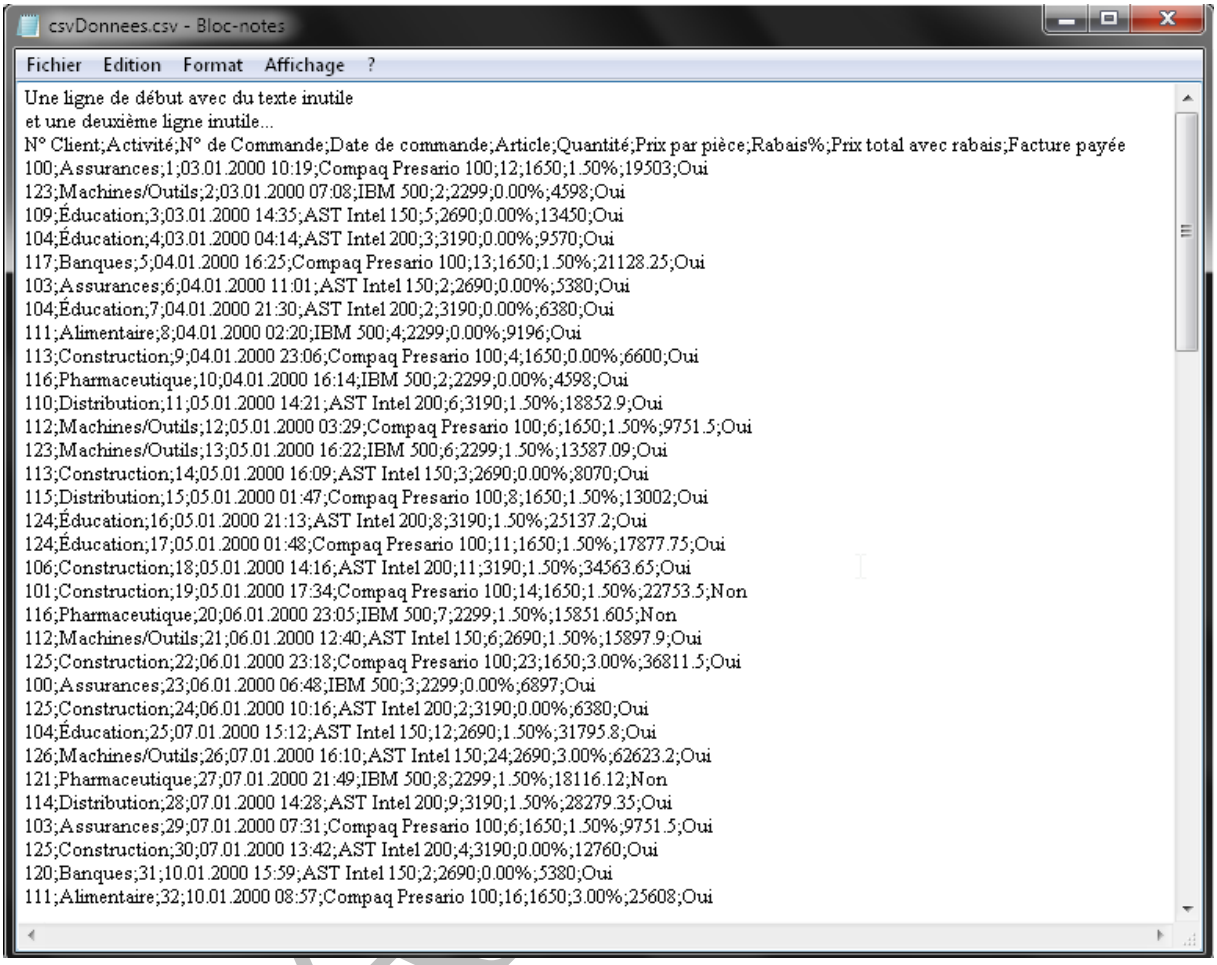


The technic is the same for cell arrays. Here is a small example as refresh:

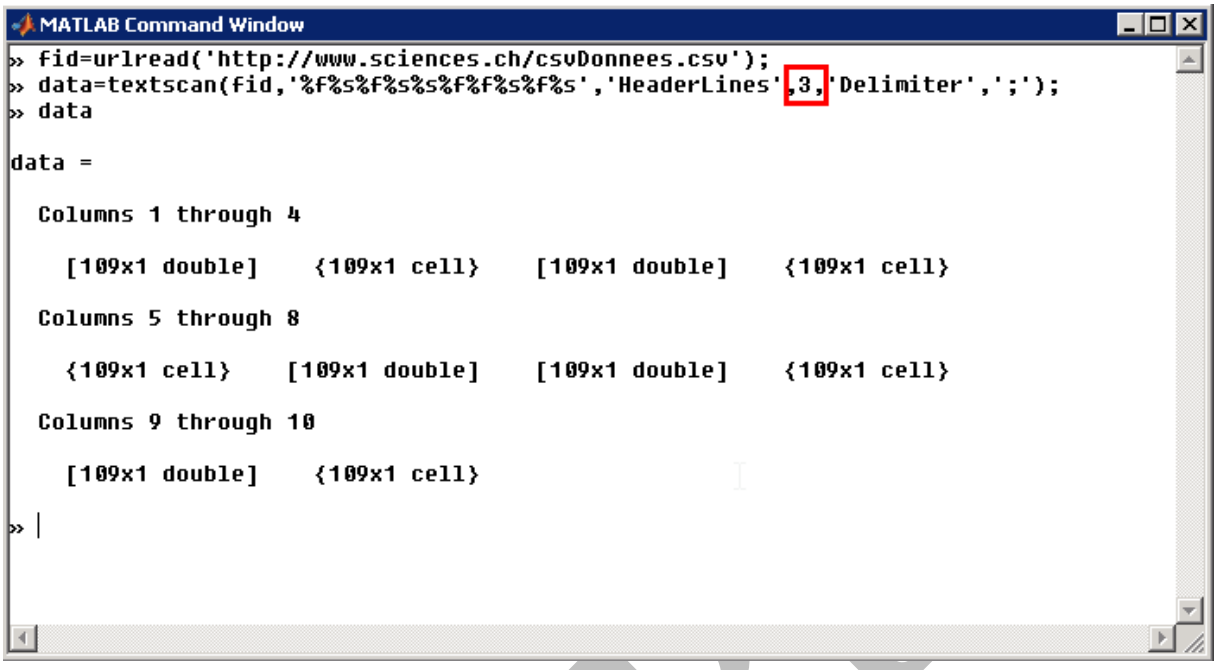


Import web csv files (with semi-colons, with headers, with text, with dates)

One very common task in academics labs is to import csv files from an URL. Suppose we have the following file on an external web server:



To import it we just use the function **urlread( )** associated to the function **textscan( )** and we get a cell array:



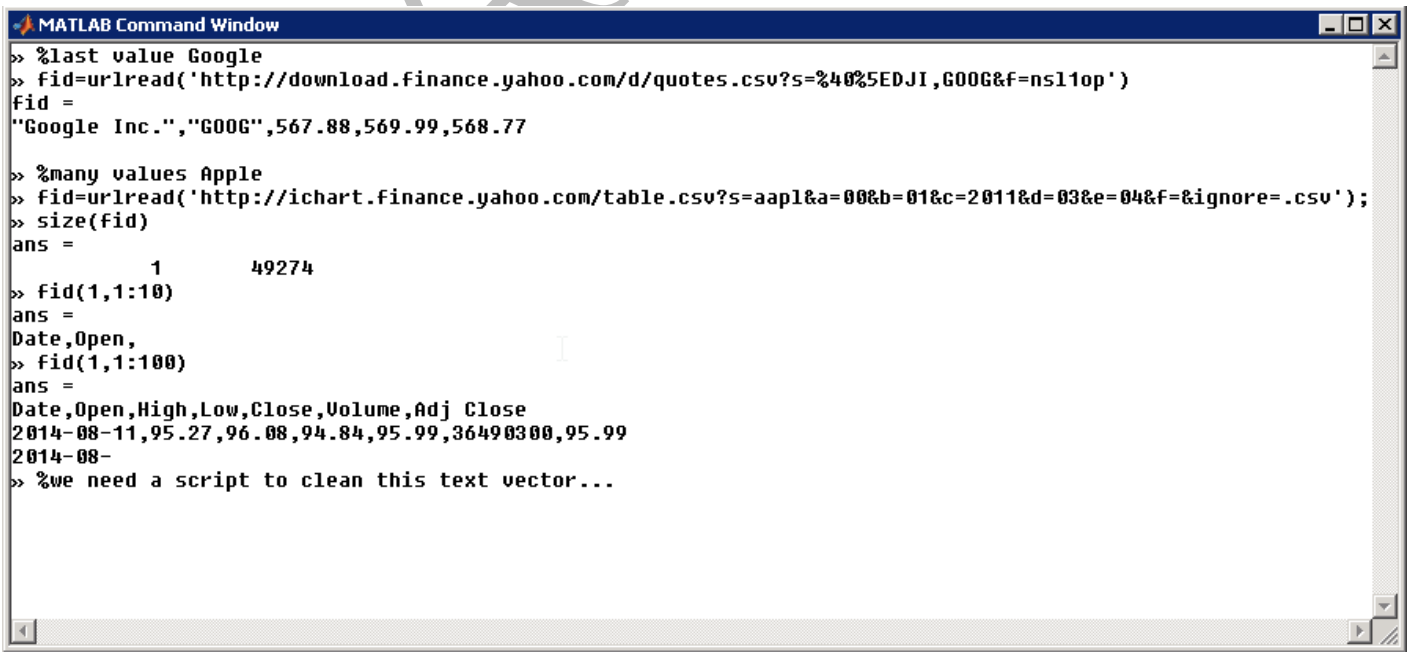
Import from Yahoo Finance

Without scripting or without Toolboxes (especially the MathWorks Datafeed Toolbox) importing data from yahoo finance is annoying.

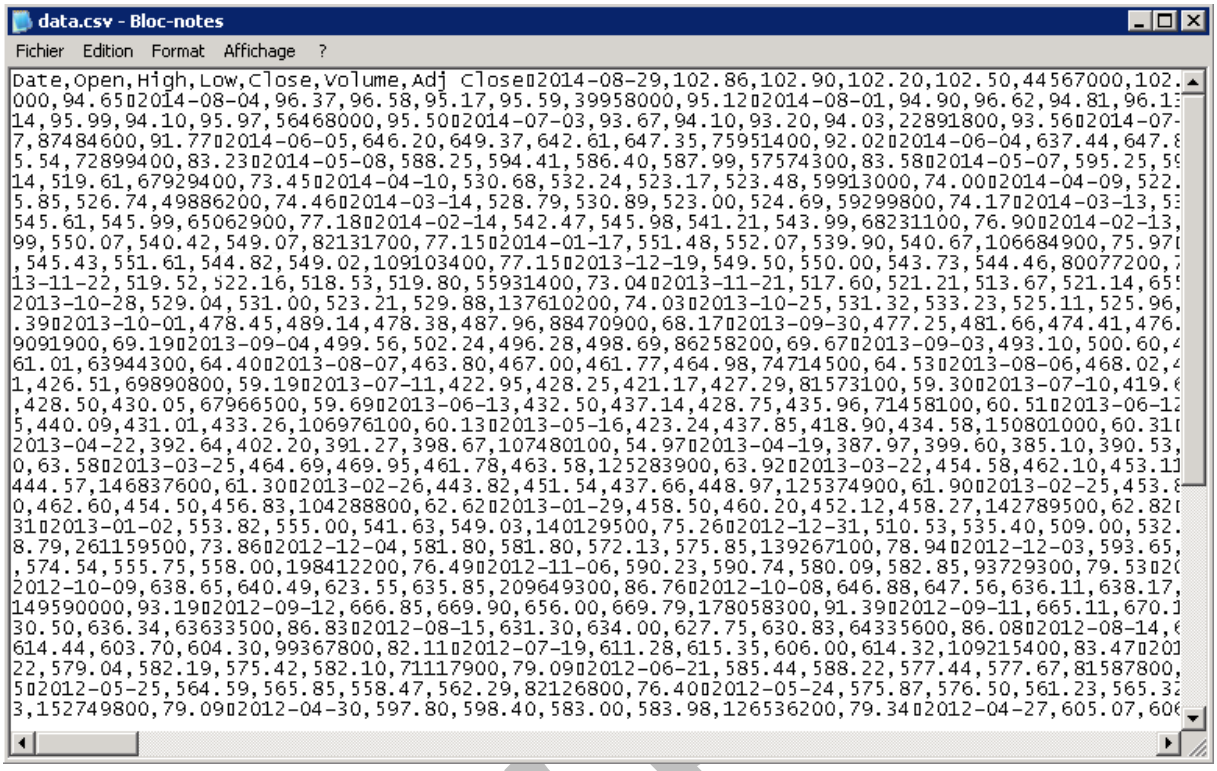
First see some descriptions:

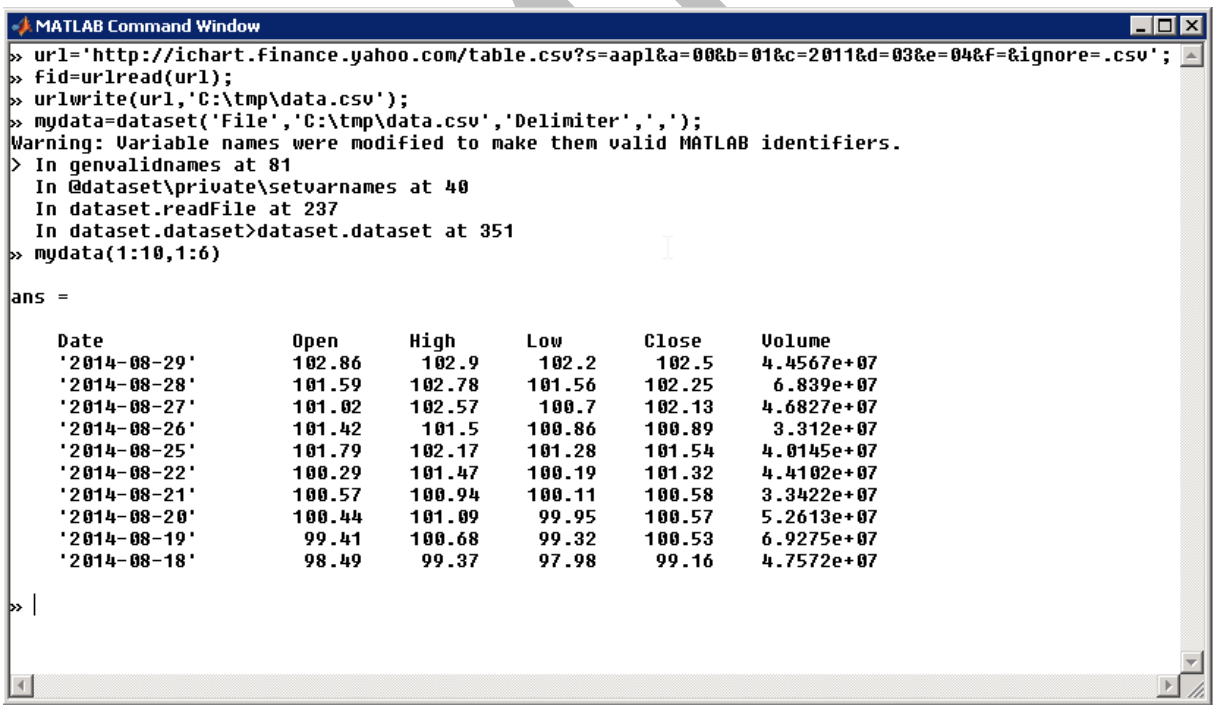
https://code.google.com/p/yahoo-finance-managed/wiki/csvQuotesDownload

And here is an example:

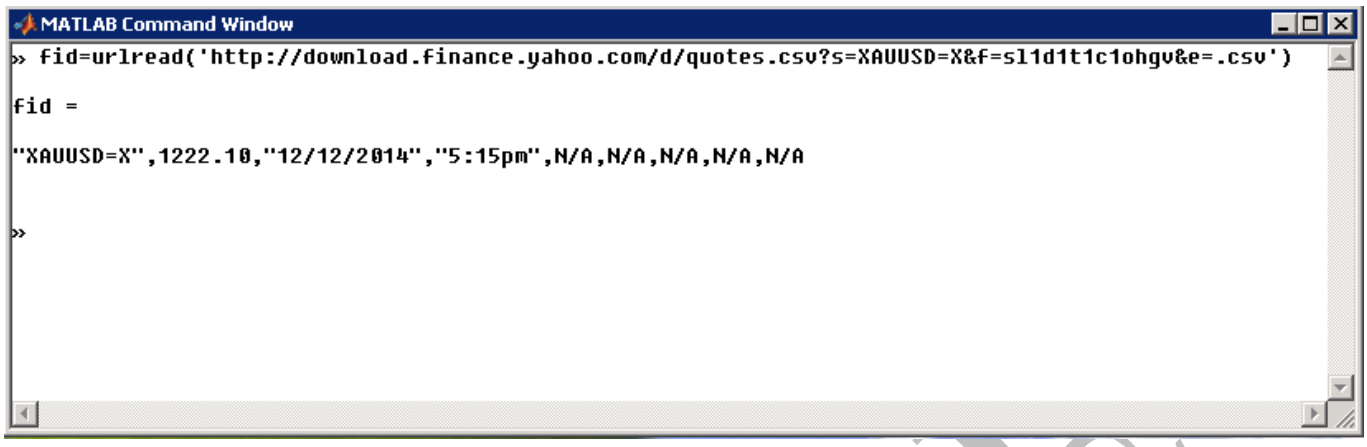


Here is the most esthetic way I have found until today for this purpose:





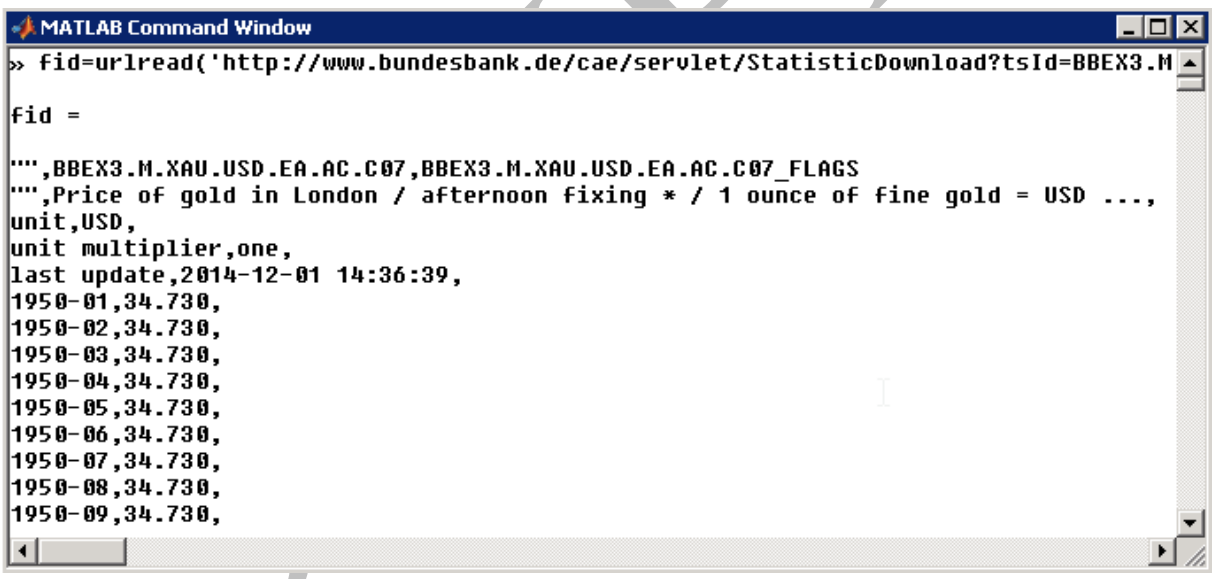
Otherwise to get last communicate Gold value from Yahoo in US$:



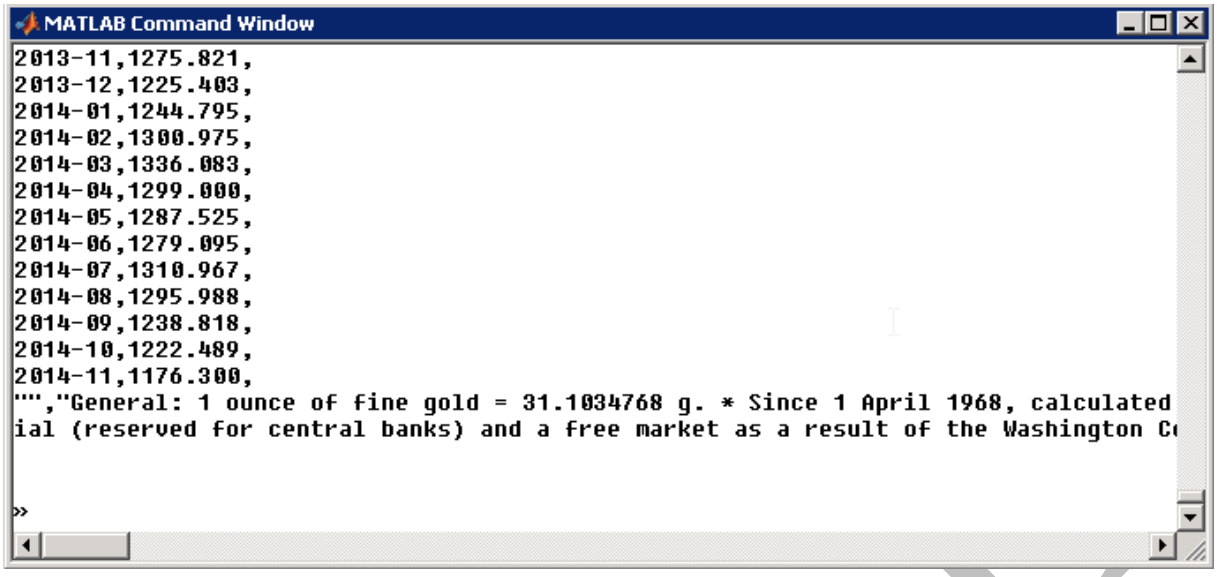
Or better (it is not very hard) using the value from the German BundesBank (the link can be found easily on Quandl.com)

http://www.bundesbank.de/cae/servlet/StatisticDownload?tsId=BBEX3.M.XAU.USD.EA.AC.C07&its_csvFormat=en&its_fileFormat=csv&mode=its

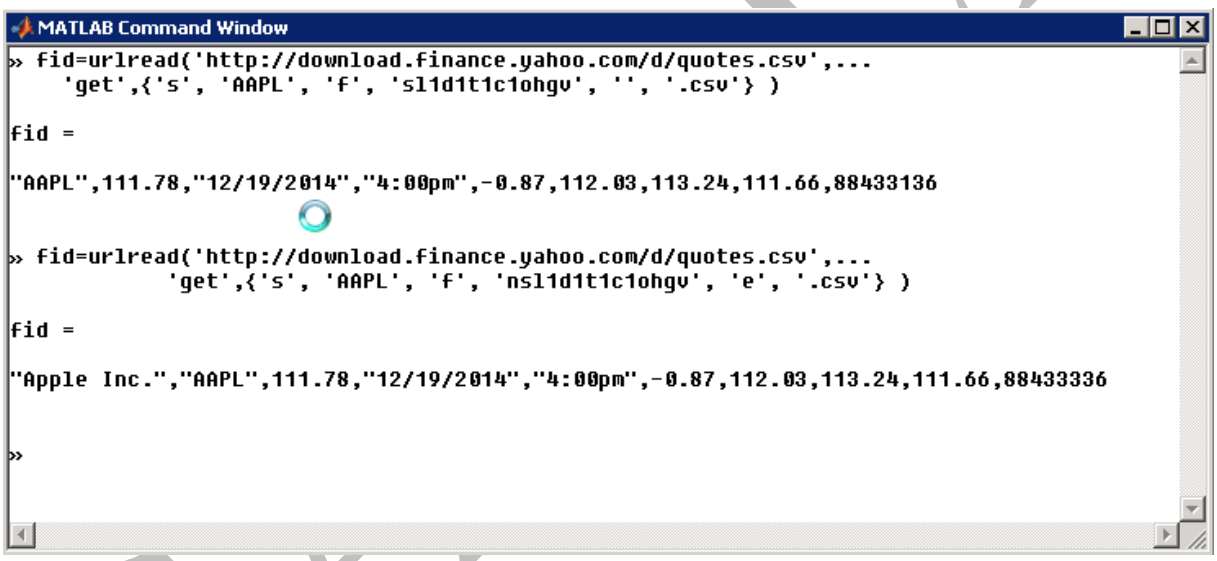
You get Gold values from:



To today's date:

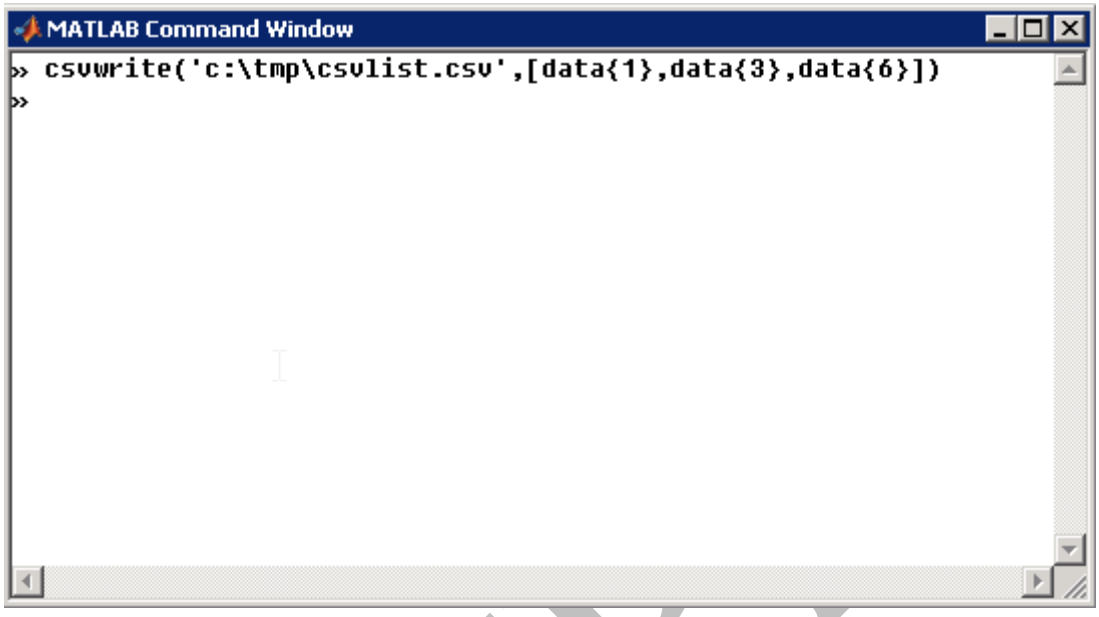


Or some others feeds:

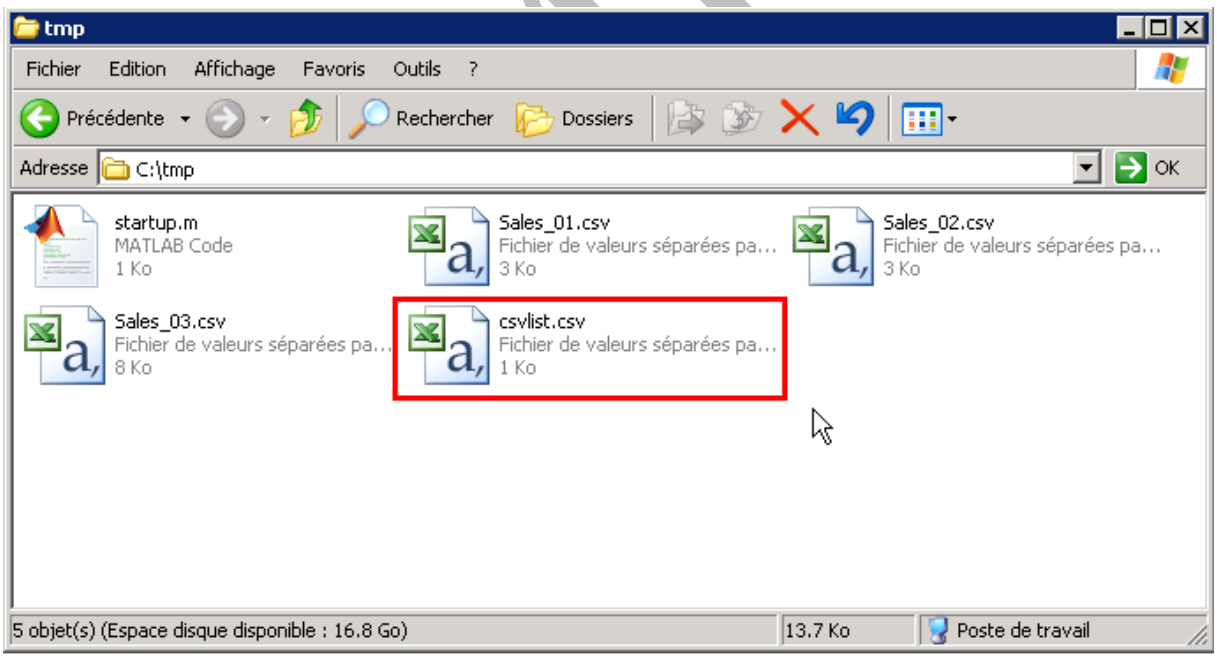


Export to csv file

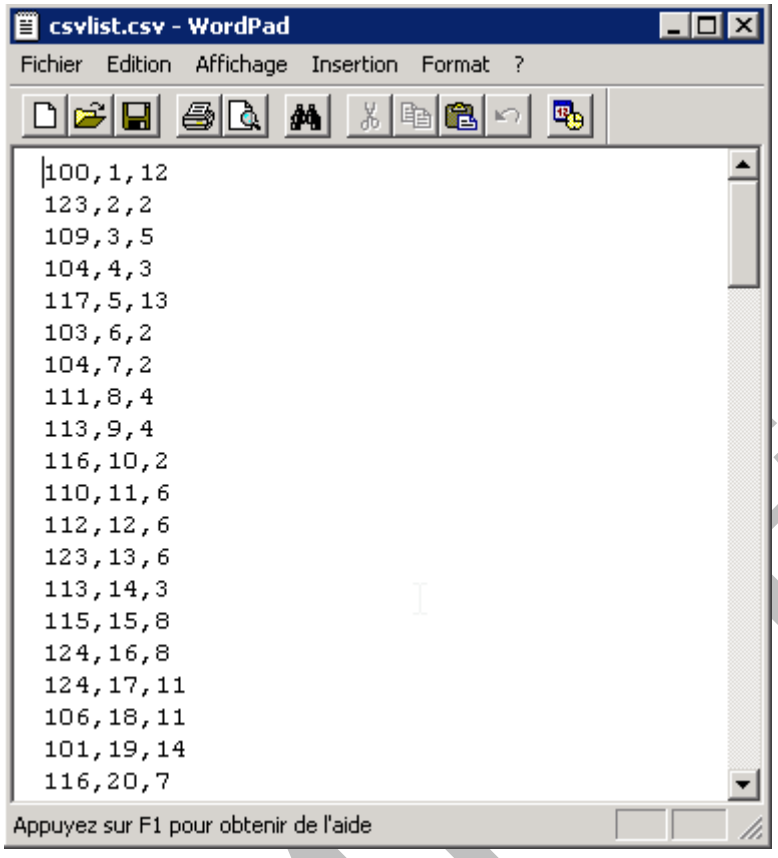
Now also something very important is to export manipulated data to a *.csv file. This can be done easily by using the function **csvwrite( )**:



The file will then be created:



And we will have the correct content:


csvlist.csv - WordPad
Fichier Edition Affichage Insertion Format ?
100,1,12
123,2,2
109,3,5
104,4,3
117,5,13
103,6,2
104,7,2
111,8,4
113,9,4
116,10,2
110,11,6
112,12,6
123,13,6
113,14,3
115,15,8
124,16,8
124,17,11
106,18,11
101,19,14
116,20,7
Appuyez sur F1 pour obtenir de l'aide

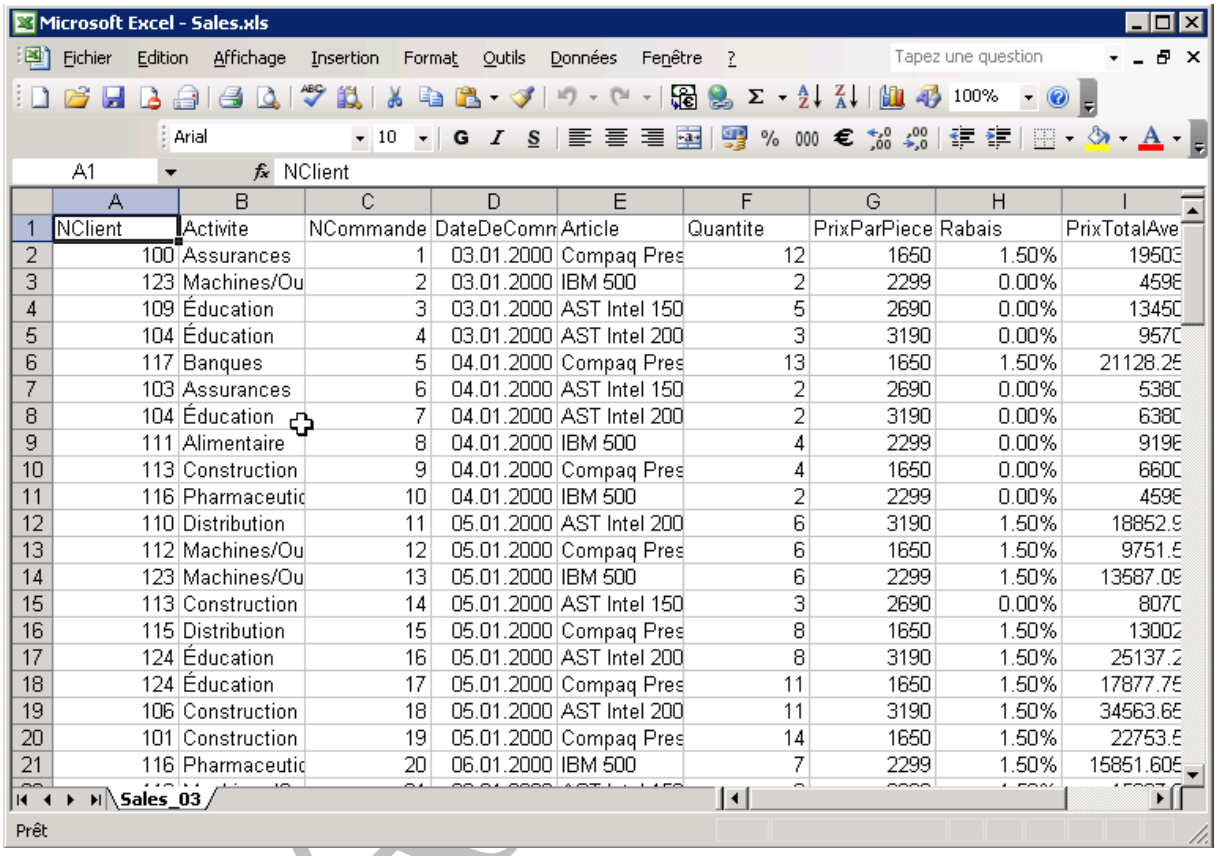
SAMPLE

Import/Export xls/xlsx (Microsoft Office Excel) files

We will see here how to import *.xls or *.xlsx files without using the import wizard of MATLAB but just by using the *Command Window*.

Import xls/xlsx local files

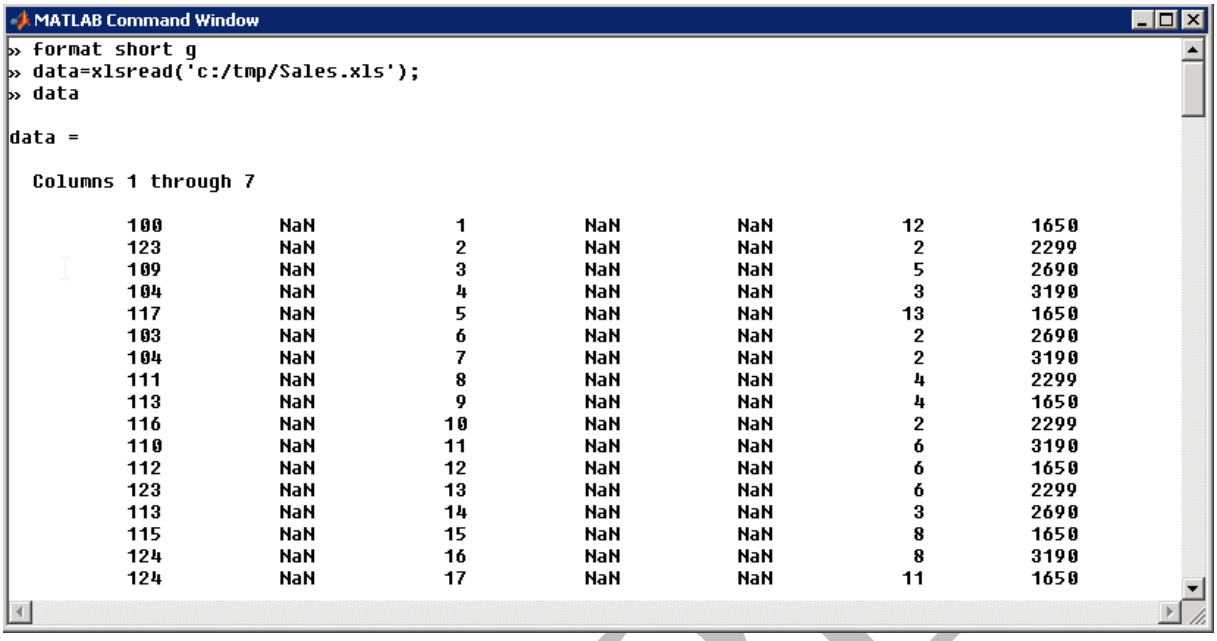
Suppose we have the following Microsoft Excel file:



	A	B	C	D	E	F	G	H	I
1	NClient	Activite	NCommande	DateDeComm	Article	Quantite	PrixParPiece	Rabais	PrixTotalAve
2	100	Assurances	1	03.01.2000	Compaq Pres	12	1650	1.50%	19503
3	123	Machines/Ou	2	03.01.2000	IBM 500	2	2299	0.00%	4598
4	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0.00%	13450
5	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0.00%	9570
6	117	Banques	5	04.01.2000	Compaq Pres	13	1650	1.50%	21128.25
7	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0.00%	5380
8	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0.00%	6380
9	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0.00%	9196
10	113	Construction	9	04.01.2000	Compaq Pres	4	1650	0.00%	6600
11	116	Pharmaceutic	10	04.01.2000	IBM 500	2	2299	0.00%	4598
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	1.50%	18852.9
13	112	Machines/Ou	12	05.01.2000	Compaq Pres	6	1650	1.50%	9751.5
14	123	Machines/Ou	13	05.01.2000	IBM 500	6	2299	1.50%	13587.09
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0.00%	8070
16	115	Distribution	15	05.01.2000	Compaq Pres	8	1650	1.50%	13002
17	124	Éducation	16	05.01.2000	AST Intel 200	8	3190	1.50%	25137.2
18	124	Éducation	17	05.01.2000	Compaq Pres	11	1650	1.50%	17877.75
19	106	Construction	18	05.01.2000	AST Intel 200	11	3190	1.50%	34563.65
20	101	Construction	19	05.01.2000	Compaq Pres	14	1650	1.50%	22753.5
21	116	Pharmaceutic	20	06.01.2000	IBM 500	7	2299	1.50%	15851.605

SA

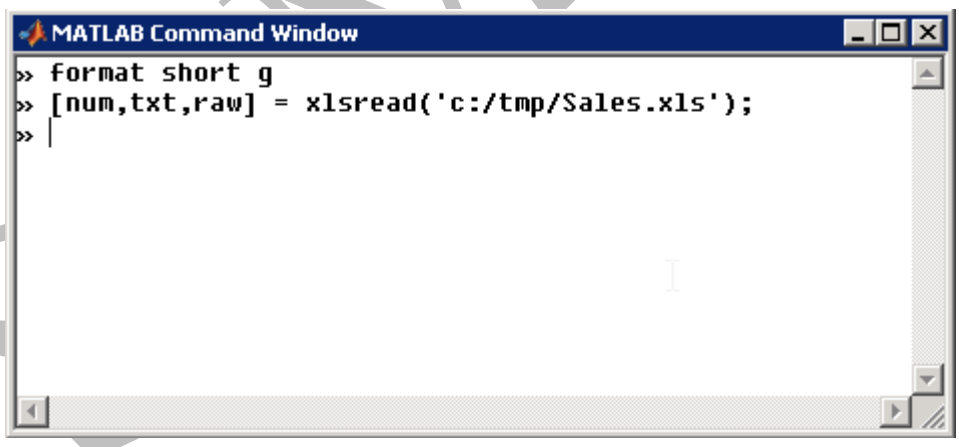
To import simple the content but without titles and string columns we can simply use the **xlsread( )** function:

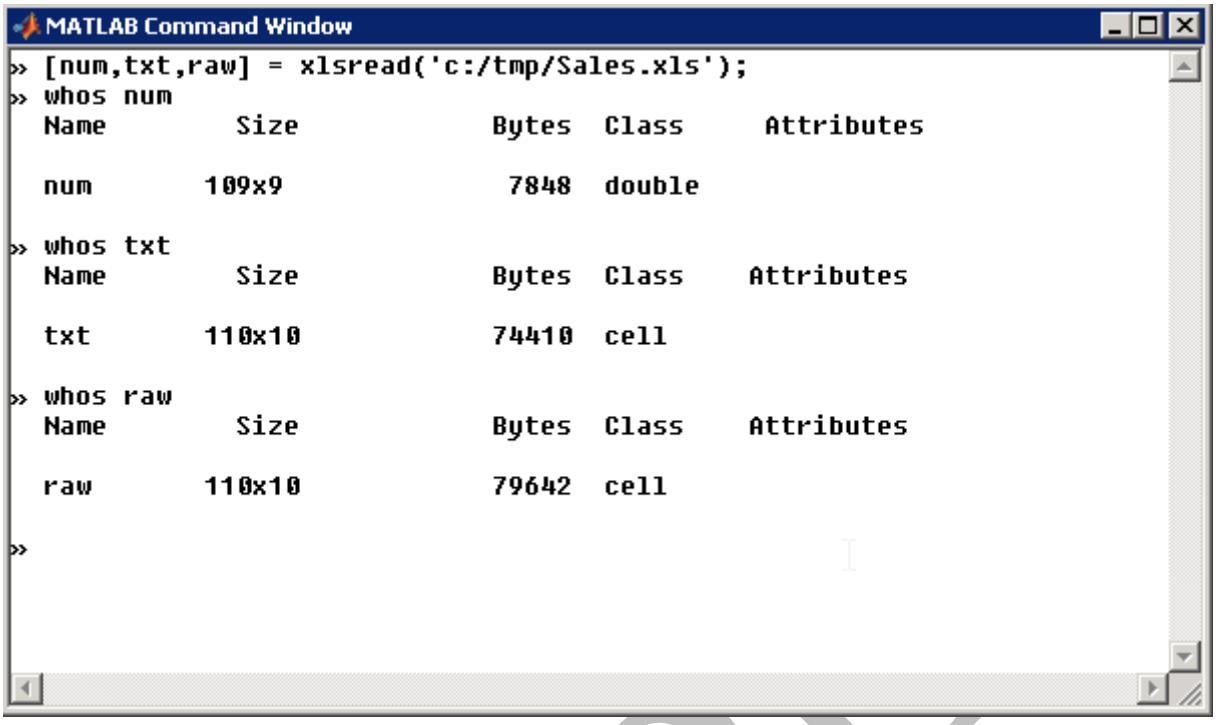


If necessary you can specify the sheet to read by writing:

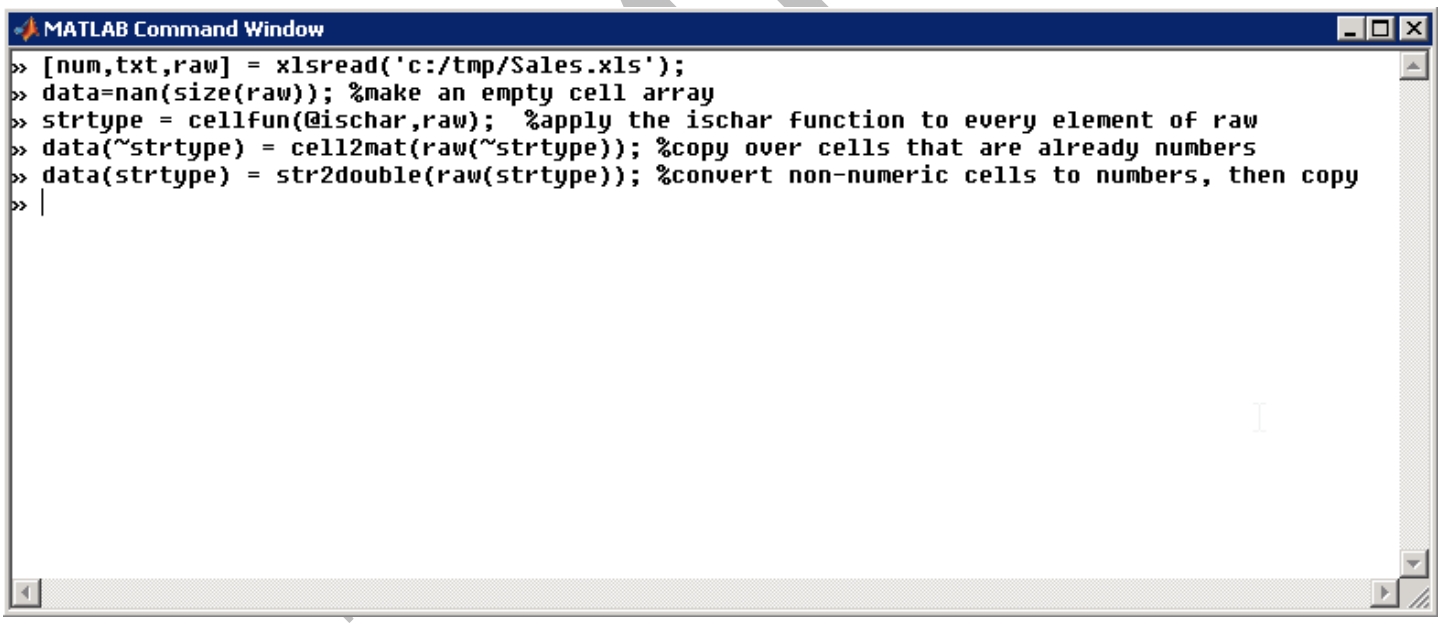
myData = xlsread('c:/tmp/Sales.xls','Sheet1').

This is an issue... but in fact there is a workaround because the function **xlsread( )** returns multiples outputs as you can see below:

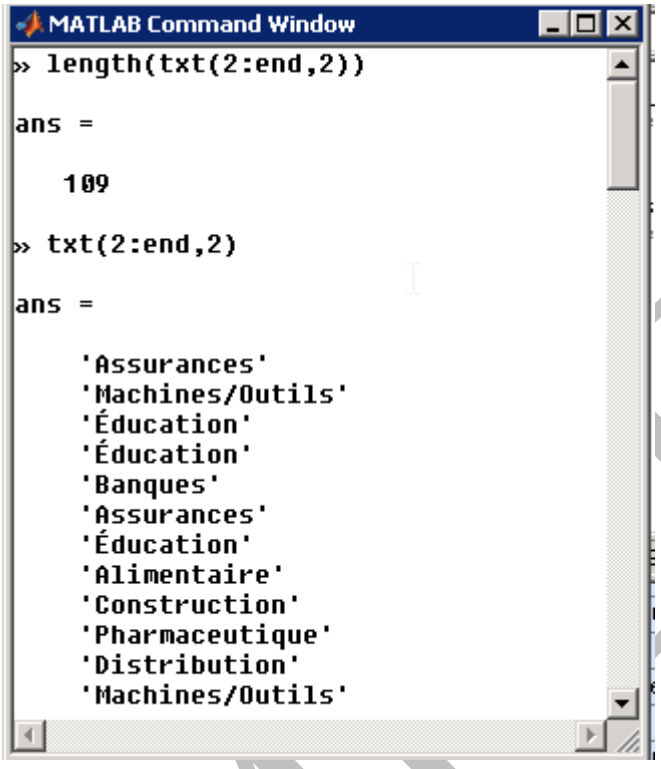




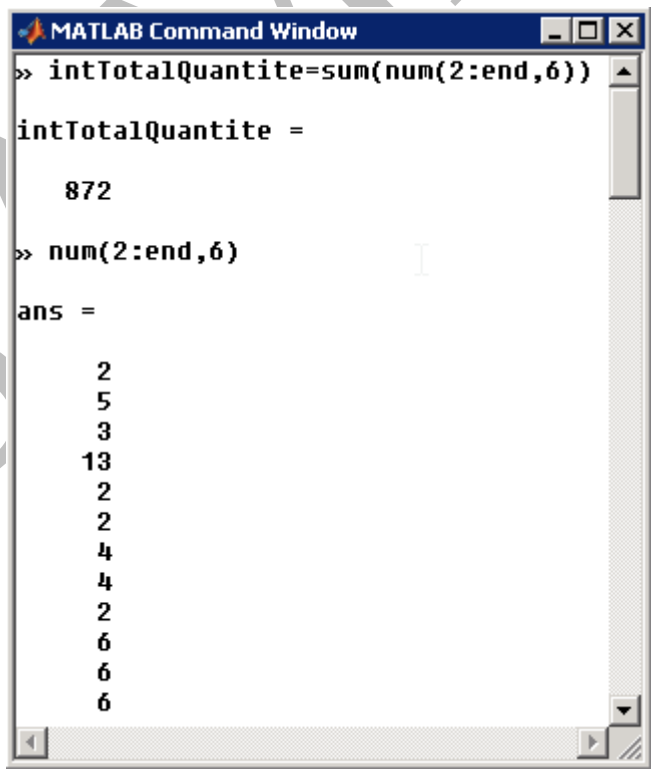
Now we will use some tricks that are not very intuitive to merge quickly everyhting in a single cell array:



Otherwise you can access to the content of the *text* cell array if you don't want to clean the data:



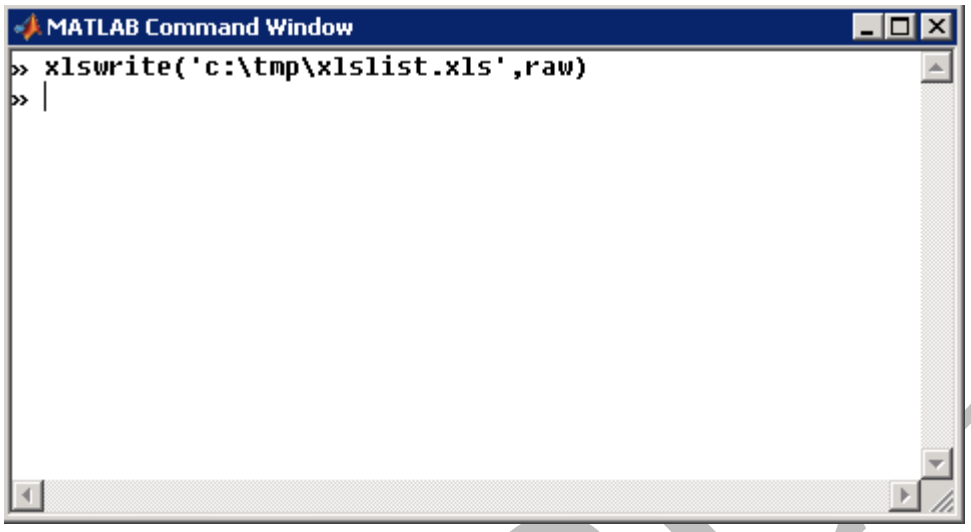
Or directly to the *num* content for calculations:



SAMPLE

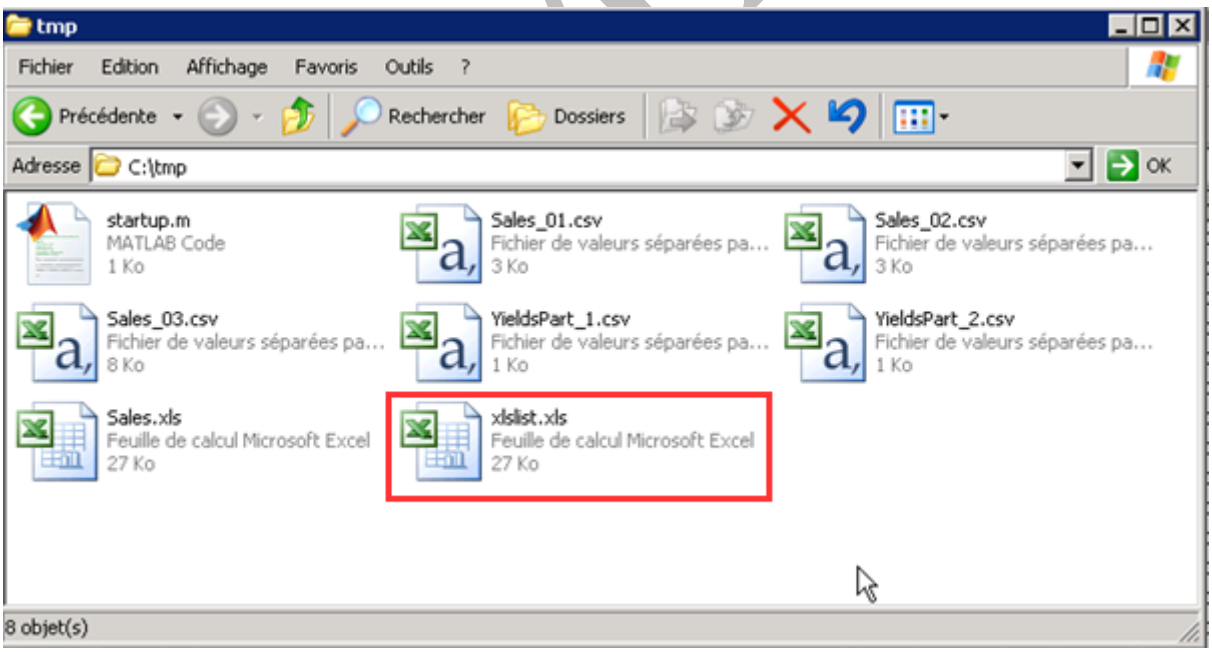
Export to xls/xlsx files

Now also something very important is to export manipulated data to a *.csv file. This can be done easily by using the function **xlswrite( )**:

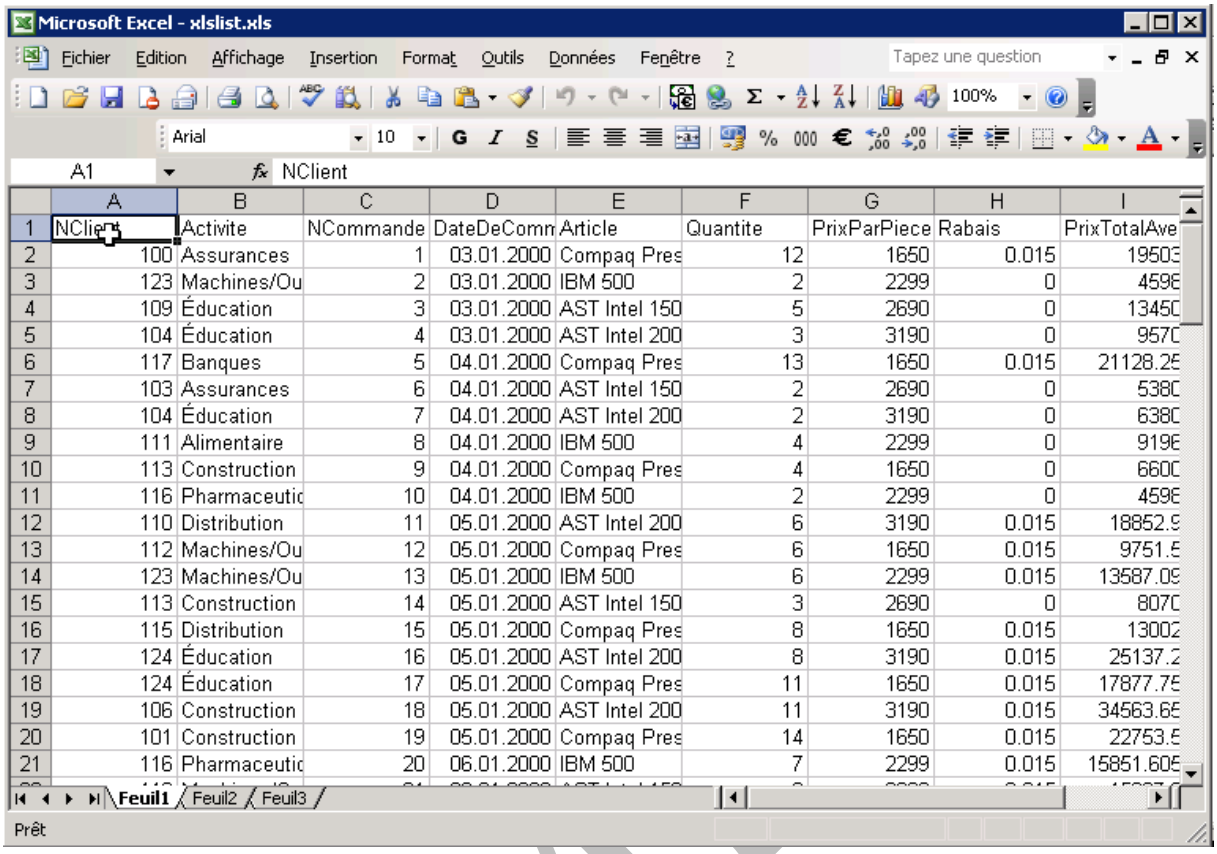


Note: To write in a specific sheet in a specific range just write:

The file will then be created:



And we will have the correct content:



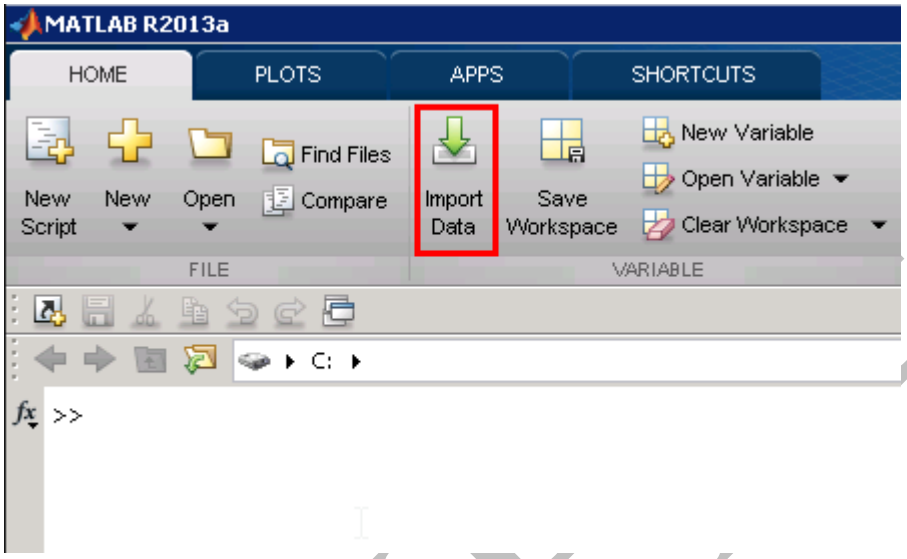
	A	B	C	D	E	F	G	H	I
1	NClient	Activite	NCommande	DateDeComm	Article	Quantite	PrixParPiece	Rabais	PrixTotalAve
2	100	Assurances	1	03.01.2000	Compaq Pres	12	1650	0.015	19503
3	123	Machines/Ou	2	03.01.2000	IBM 500	2	2299	0	4598
4	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0	13450
5	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0	9570
6	117	Banques	5	04.01.2000	Compaq Pres	13	1650	0.015	21128.25
7	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0	5380
8	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0	6380
9	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0	9196
10	113	Construction	9	04.01.2000	Compaq Pres	4	1650	0	6600
11	116	Pharmaceutic	10	04.01.2000	IBM 500	2	2299	0	4598
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	0.015	18852.9
13	112	Machines/Ou	12	05.01.2000	Compaq Pres	6	1650	0.015	9751.5
14	123	Machines/Ou	13	05.01.2000	IBM 500	6	2299	0.015	13587.09
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0	8070
16	115	Distribution	15	05.01.2000	Compaq Pres	8	1650	0.015	13002
17	124	Éducation	16	05.01.2000	AST Intel 200	8	3190	0.015	25137.2
18	124	Éducation	17	05.01.2000	Compaq Pres	11	1650	0.015	17877.75
19	106	Construction	18	05.01.2000	AST Intel 200	11	3190	0.015	34563.65
20	101	Construction	19	05.01.2000	Compaq Pres	14	1650	0.015	22753.5
21	116	Pharmaceutic	20	06.01.2000	IBM 500	7	2299	0.015	15851.605

SAMPLE

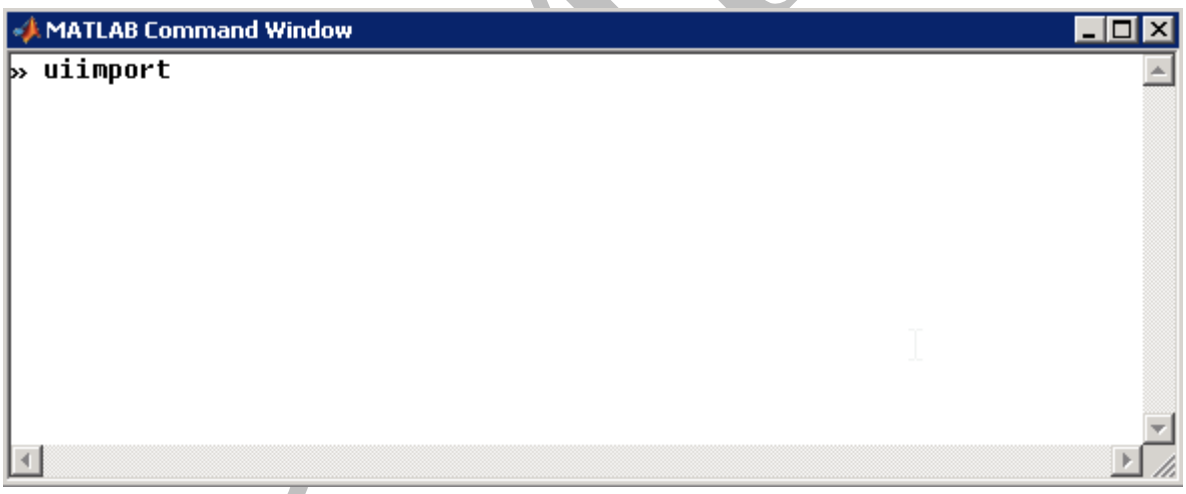
Import data with the wizard

There are some people that don't like to use the terminal to import data, they prefer to import using a WYSIWYG wizard.

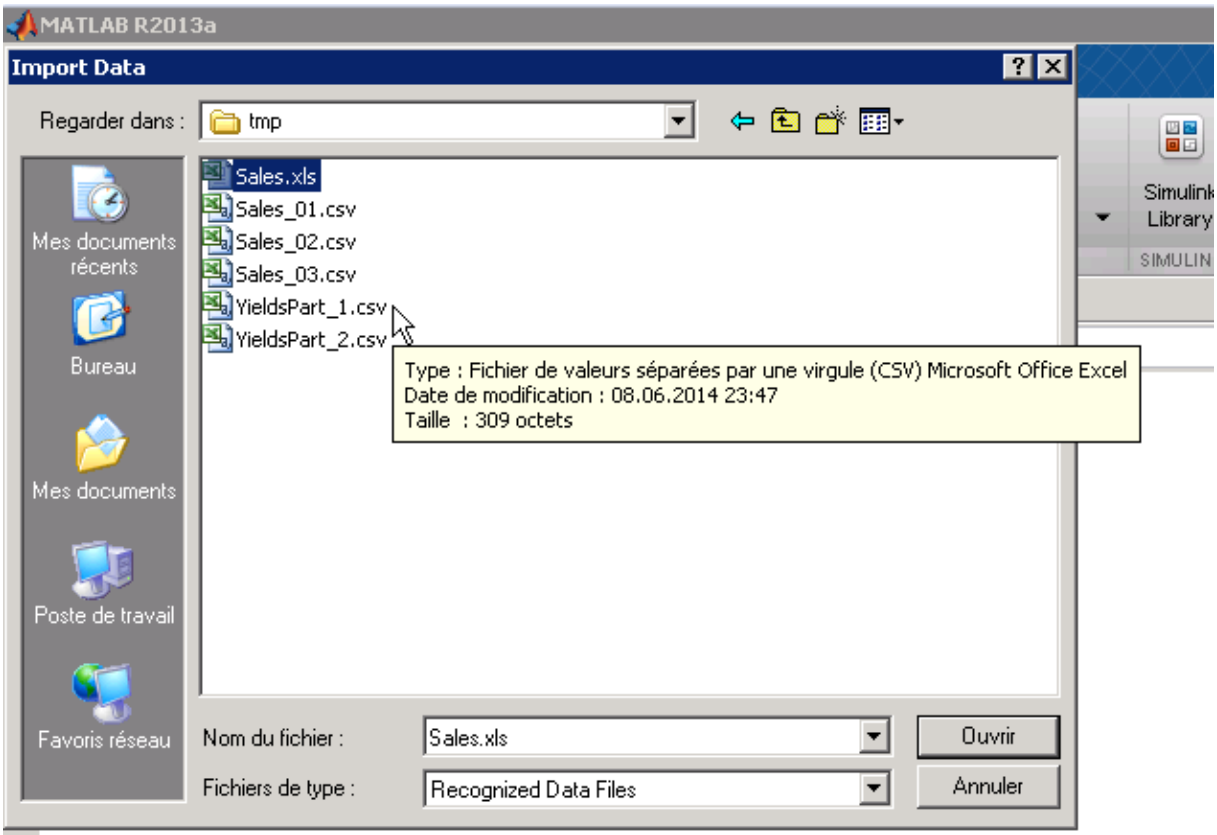
Thus in the **Home** ribbon click on the button **Import Data**:



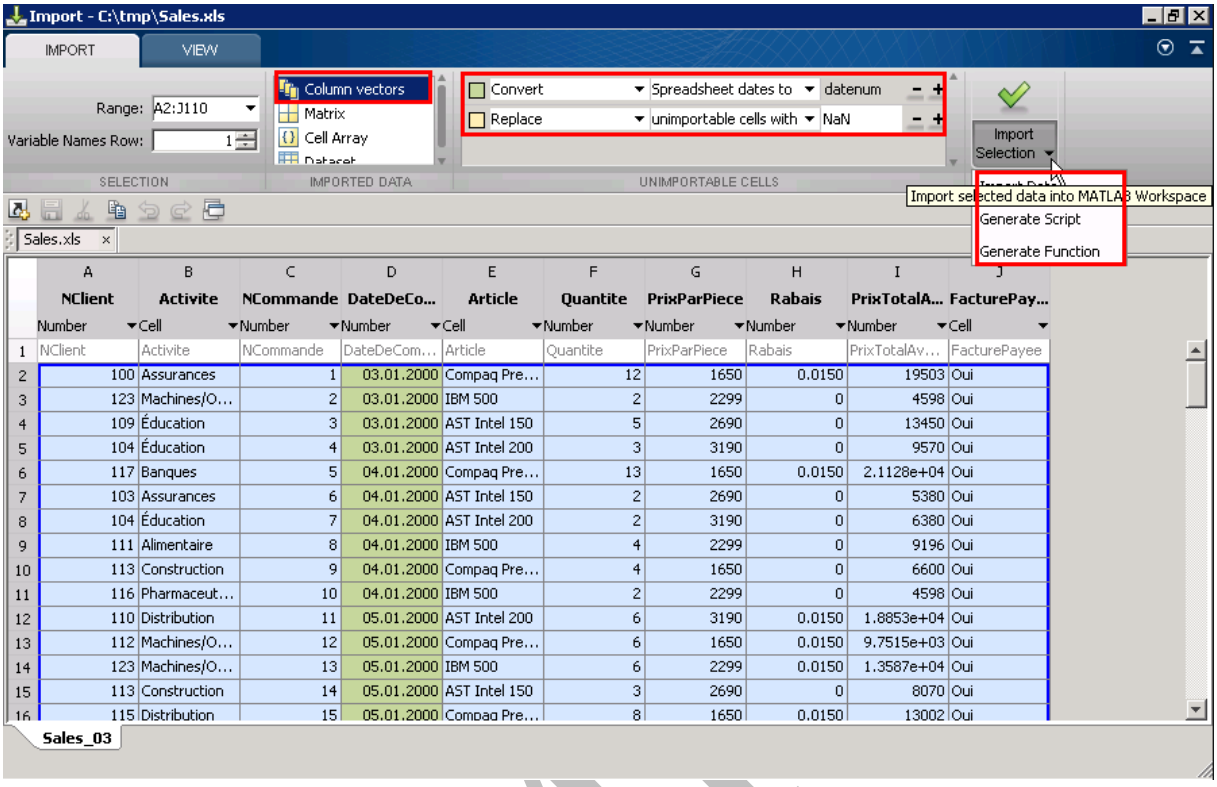
Or if you want to start this wizard from the terminal use the command **uiimport**:



When the wizard start the first thing it ask you is to choose the file with the datas you want to import:

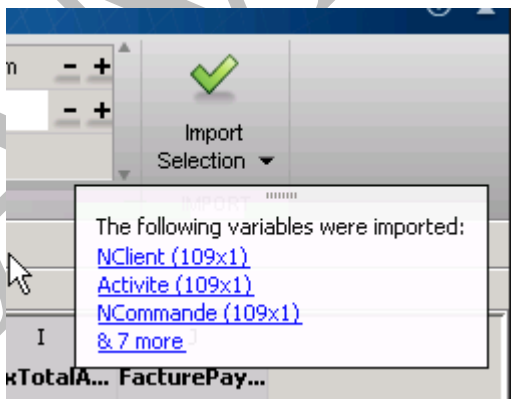


Once the data selected you get the **Import** Window with different option to import the selection as you can see below:



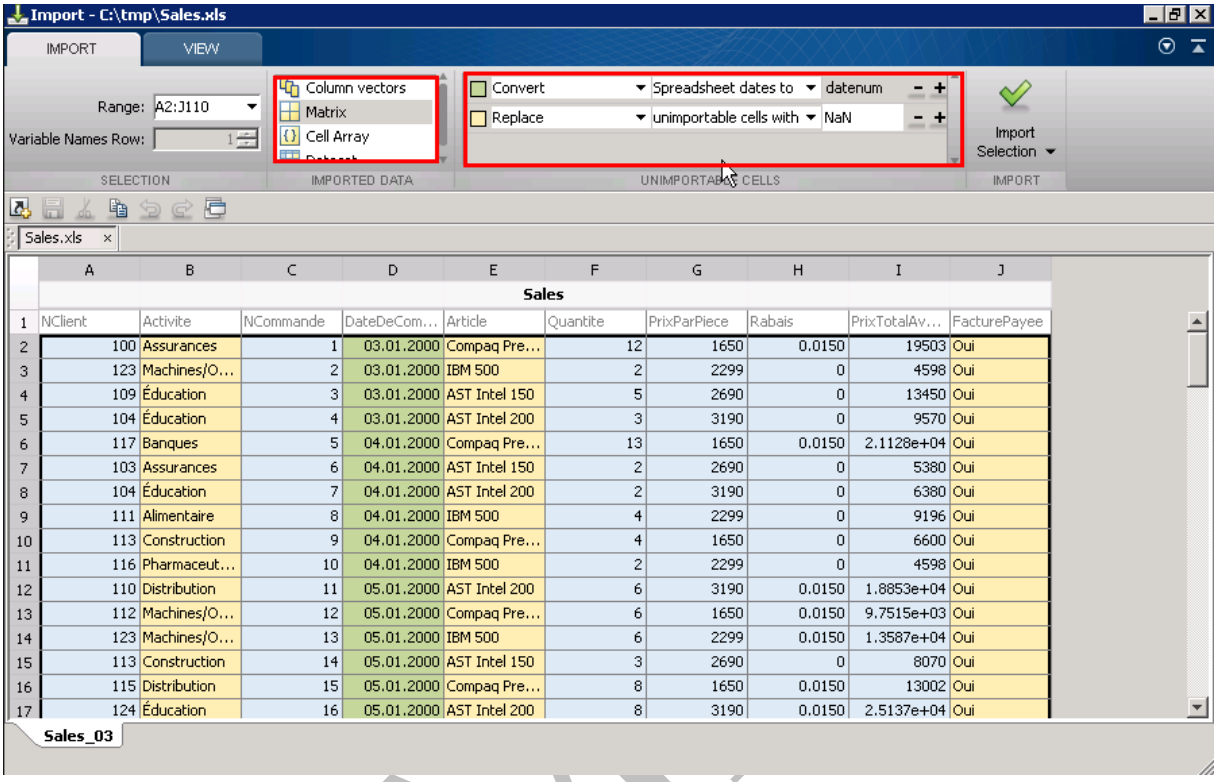
	A	B	C	D	E	F	G	H	I	J
	NClient	Activite	NCommande	DateDeCo...	Article	Quantite	PrixParPiece	Rabais	PrixTotalA...	FacturePay...
	Number	Cell	Number	Number	Cell	Number	Number	Number	Number	Cell
1	NClient	Activite	NCommande	DateDeCom...	Article	Quantite	PrixParPiece	Rabais	PrixTotalAv...	FacturePayee
2	100	Assurances	1	03.01.2000	Compaq Pre...	12	1650	0.0150	19503	Oui
3	123	Machines/O...	2	03.01.2000	IBM 500	2	2299	0	4598	Oui
4	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0	13450	Oui
5	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0	9570	Oui
6	117	Banques	5	04.01.2000	Compaq Pre...	13	1650	0.0150	2.1128e+04	Oui
7	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0	5380	Oui
8	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0	6380	Oui
9	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0	9196	Oui
10	113	Construction	9	04.01.2000	Compaq Pre...	4	1650	0	6600	Oui
11	116	Pharmaceut...	10	04.01.2000	IBM 500	2	2299	0	4598	Oui
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	0.0150	1.8853e+04	Oui
13	112	Machines/O...	12	05.01.2000	Compaq Pre...	6	1650	0.0150	9.7515e+03	Oui
14	123	Machines/O...	13	05.01.2000	IBM 500	6	2299	0.0150	1.3587e+04	Oui
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0	8070	Oui
16	115	Distribution	15	05.01.2000	Compaq Pre...	8	1650	0.0150	13002	Oui

If you select the option **Import Data** from the button **Import Selection** you will get first a small message:



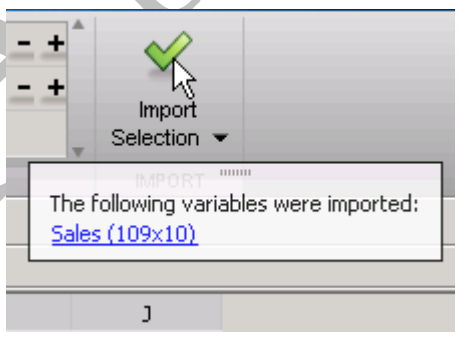
Resulting in imported 10 vectors in 10 separated variables.

See carefully that depending on the type of variable you want to import your data in (**Columm Vectors**, **Matrix**, **Cell Array** or **Dataset**) the colors in the sheet will change to inform you to what will be converted (in green) and what will be replaced (in yellow) as you can see below:



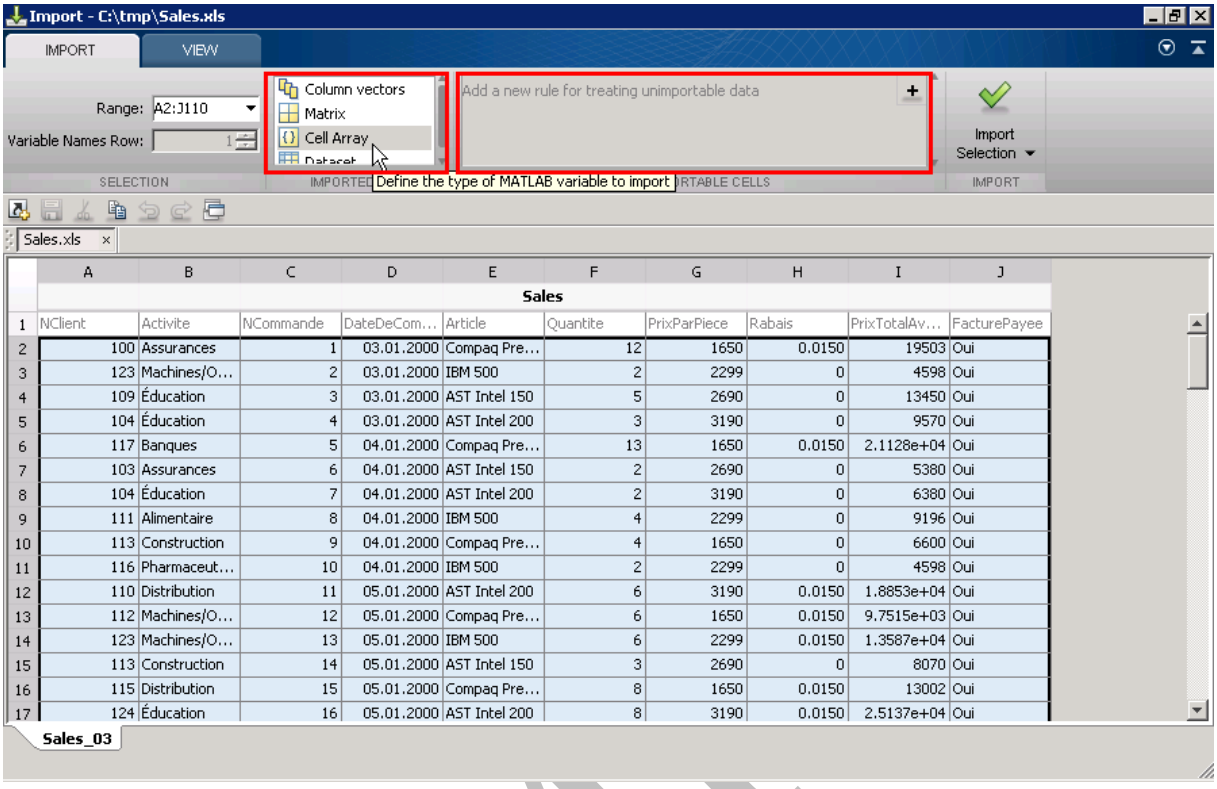
	A	B	C	D	E	F	G	H	I	J
	Sales									
1	NClient	Activite	NCommande	DateDeCom...	Article	Quantite	PrixParPiece	Rabais	PrixTotalAv...	FacturePayee
2	100	Assurances	1	03.01.2000	Compaq Pre...	12	1650	0.0150	19503	Oui
3	123	Machines/O...	2	03.01.2000	IBM 500	2	2299	0	4598	Oui
4	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0	13450	Oui
5	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0	9570	Oui
6	117	Banques	5	04.01.2000	Compaq Pre...	13	1650	0.0150	2.1128e+04	Oui
7	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0	5380	Oui
8	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0	6380	Oui
9	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0	9196	Oui
10	113	Construction	9	04.01.2000	Compaq Pre...	4	1650	0	6600	Oui
11	116	Pharmaceut...	10	04.01.2000	IBM 500	2	2299	0	4598	Oui
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	0.0150	1.8853e+04	Oui
13	112	Machines/O...	12	05.01.2000	Compaq Pre...	6	1650	0.0150	9.7515e+03	Oui
14	123	Machines/O...	13	05.01.2000	IBM 500	6	2299	0.0150	1.3587e+04	Oui
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0	8070	Oui
16	115	Distribution	15	05.01.2000	Compaq Pre...	8	1650	0.0150	13002	Oui
17	124	Éducation	16	05.01.2000	AST Intel 200	8	3190	0.0150	2.5137e+04	Oui

If you choose for example the structure **Matrix** and validate the import you will get a unique matrix with 10 columns but some columns will all be with NaN (the columns with strings that are in yellow above)

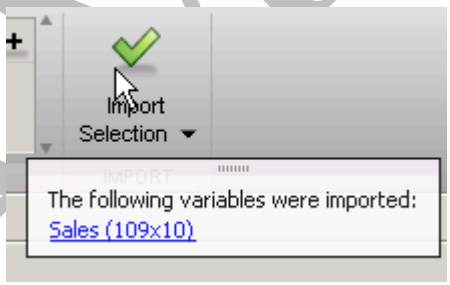


SAMPLE

If you choose **Cell Array** as structure for sure... everything can be imported as you can see it below:

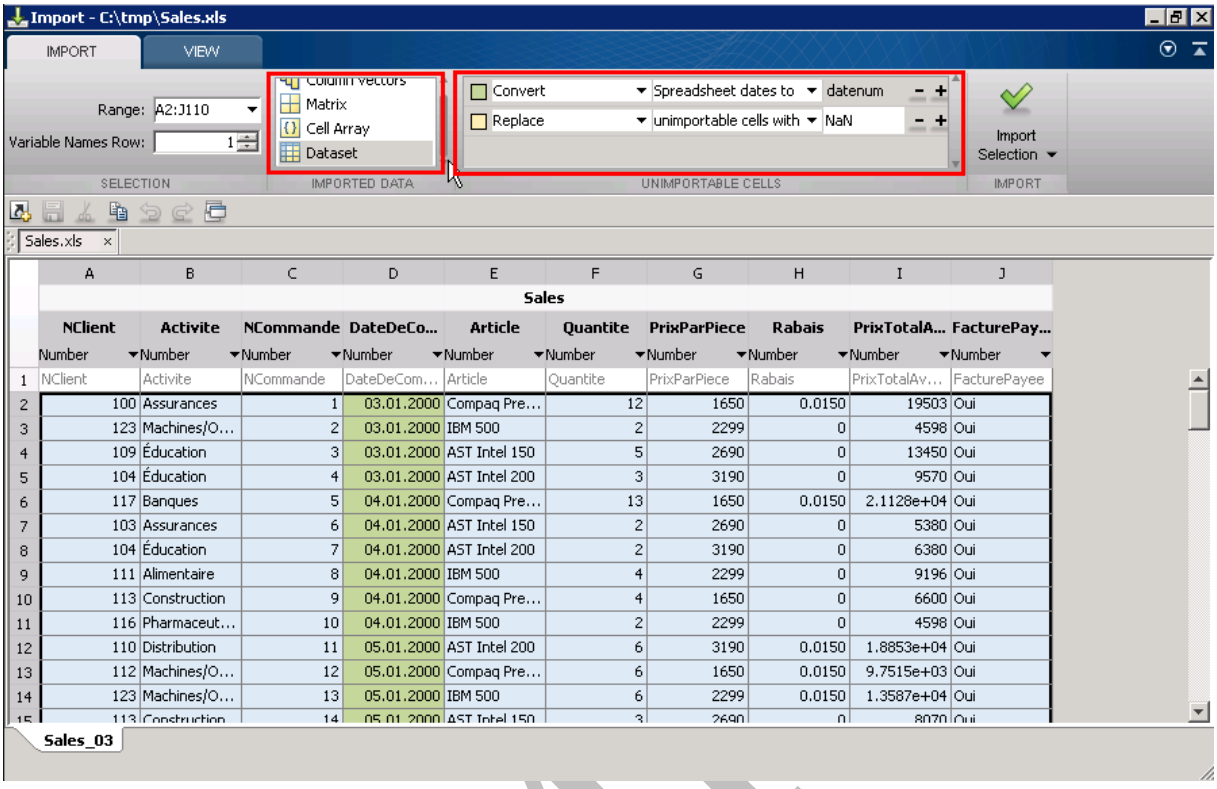


And once you validate you can see that only one variable will be created:

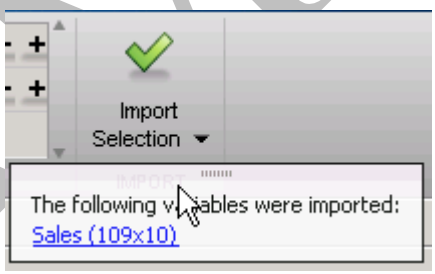


SAMPLE

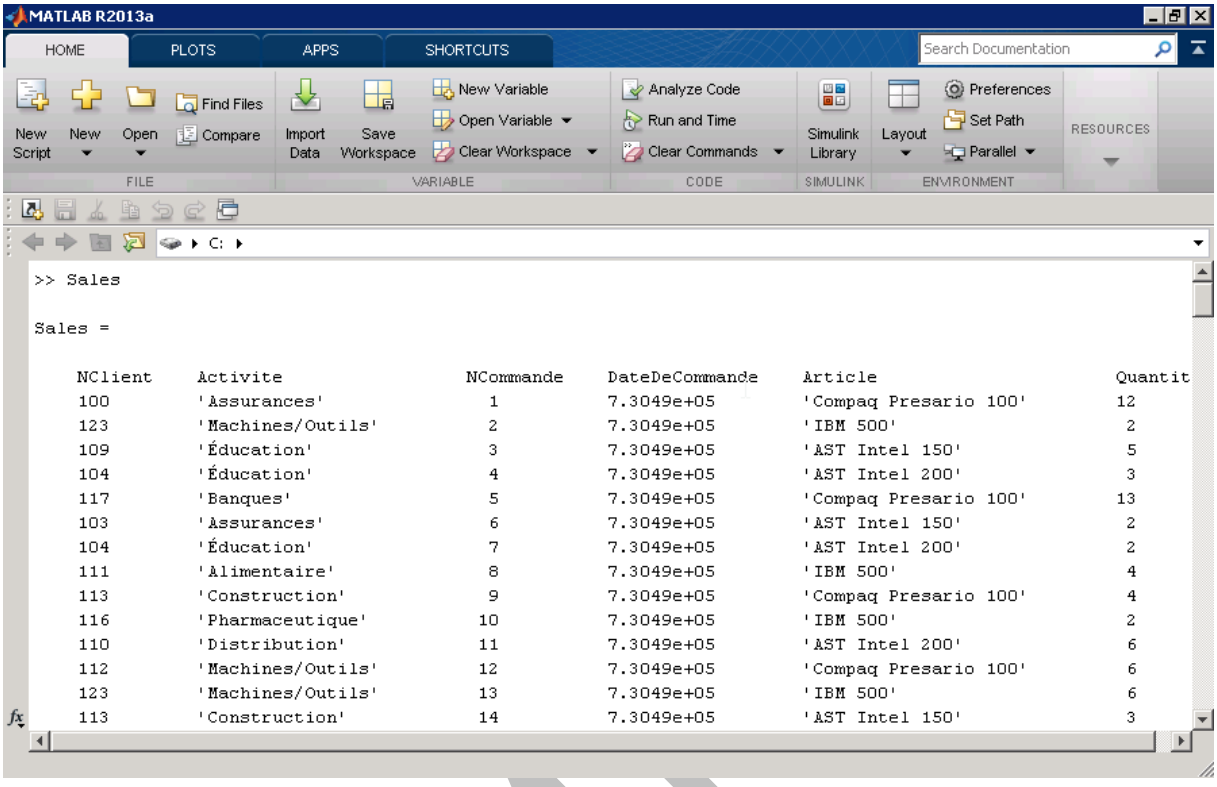
If you select **Dataset** you will not be able to import dates if you don't choose to convert them as string:



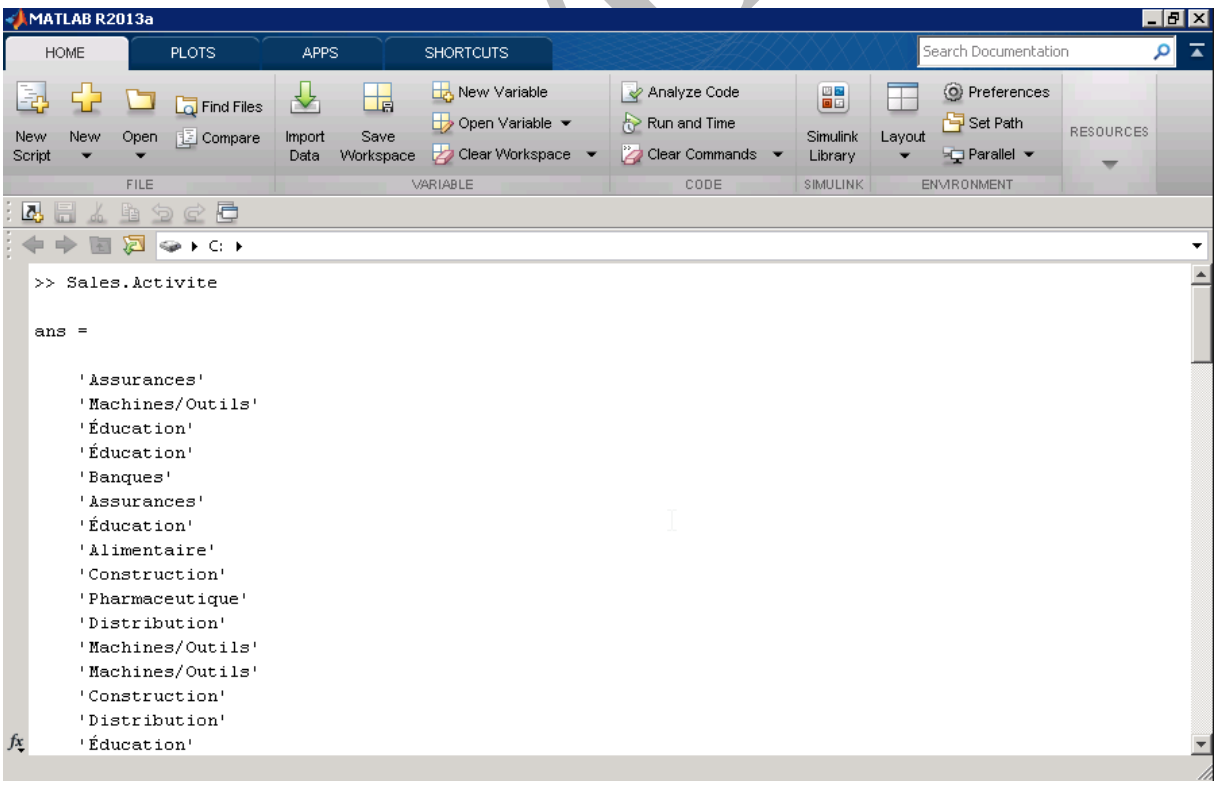
A dataset will also create only one unique variable in the workspace:



The result will looks like this when you import as a dataset:



To access to a column the syntax is very similar to R:



Charts

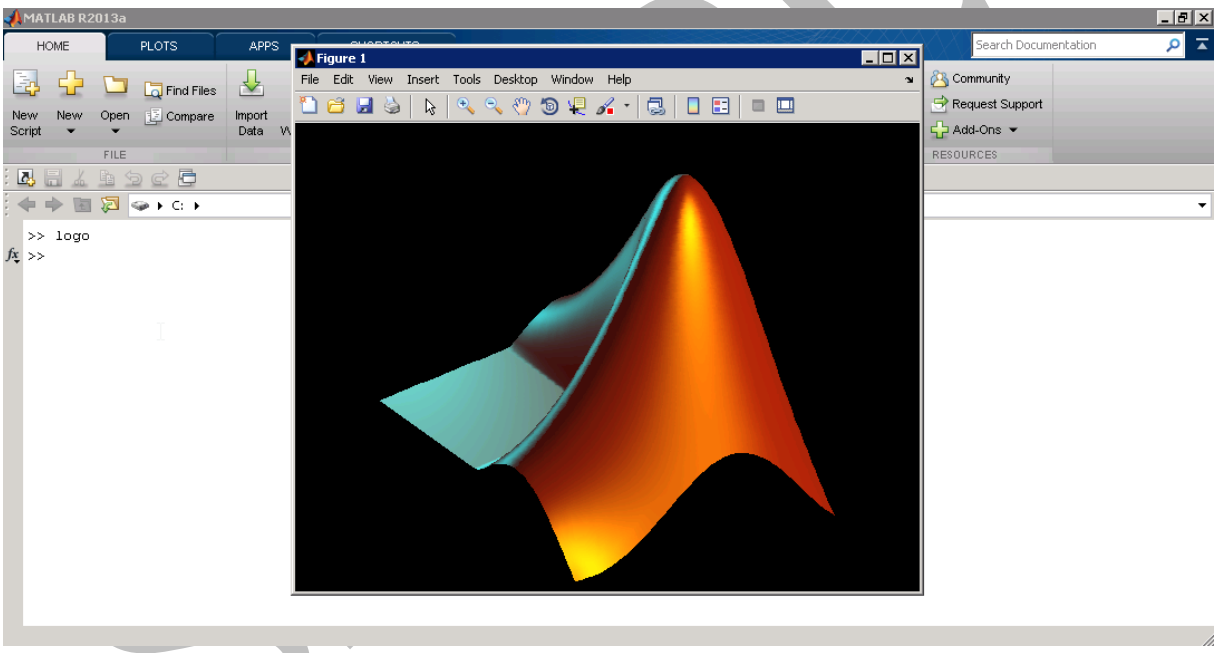
A lot of people love charts and put charts everywhere in their job. You must first remember that for sure "one image says one thousand words" but in mathematics you can apply this quote only for 1D, 2D, 3D and 4D charts but after... you have to go back to the reading of pure mathematical values.

We will see in this chapter a lot of classical, typical charts in physics, signal processing and finance that don't require any toolboxes. The mains purpose of this chapter in facts is to show a lot of "hidden" tips in MATLAB and answer to most common questions of the users following my trainings.

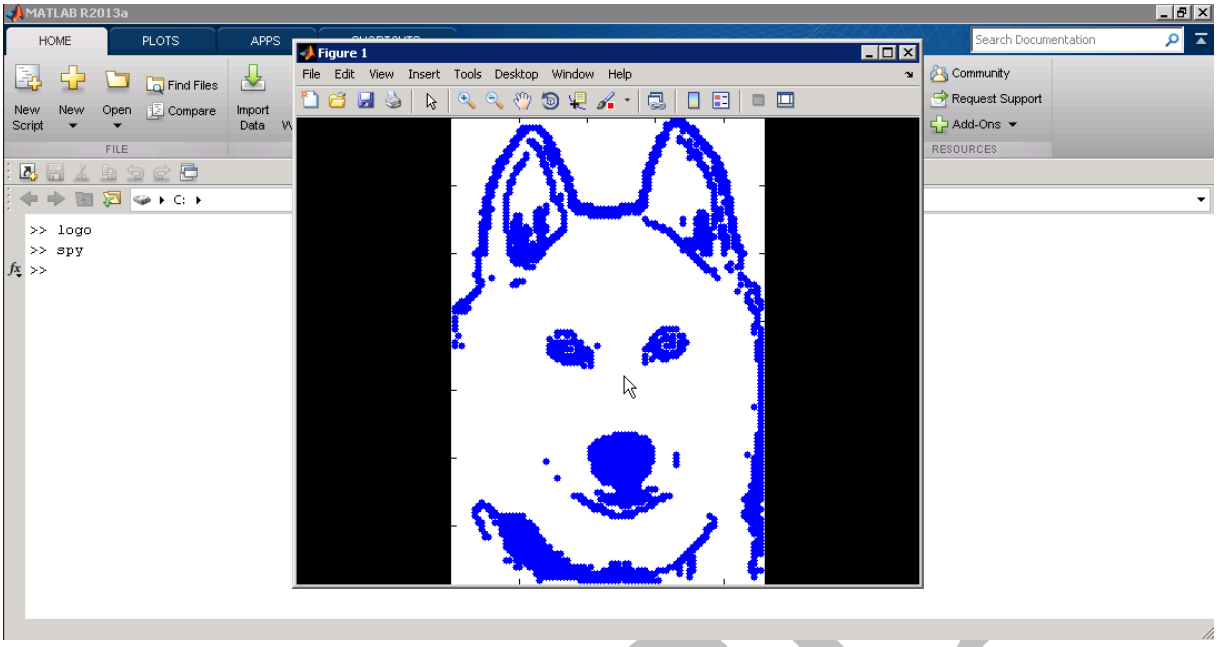
But first some fun...

Demo

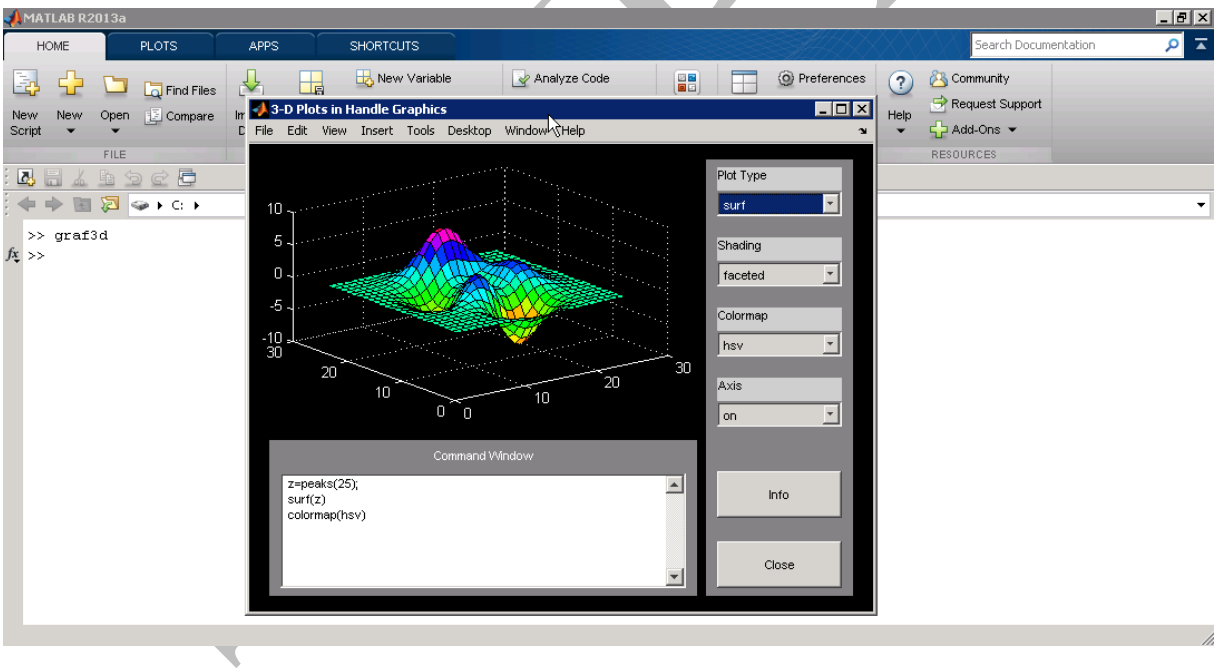
In MATLAB there are some demo charts that can be useful to learn the basics alone. For example if you the command **logo** you will get...:



And if you type the command **spy**:

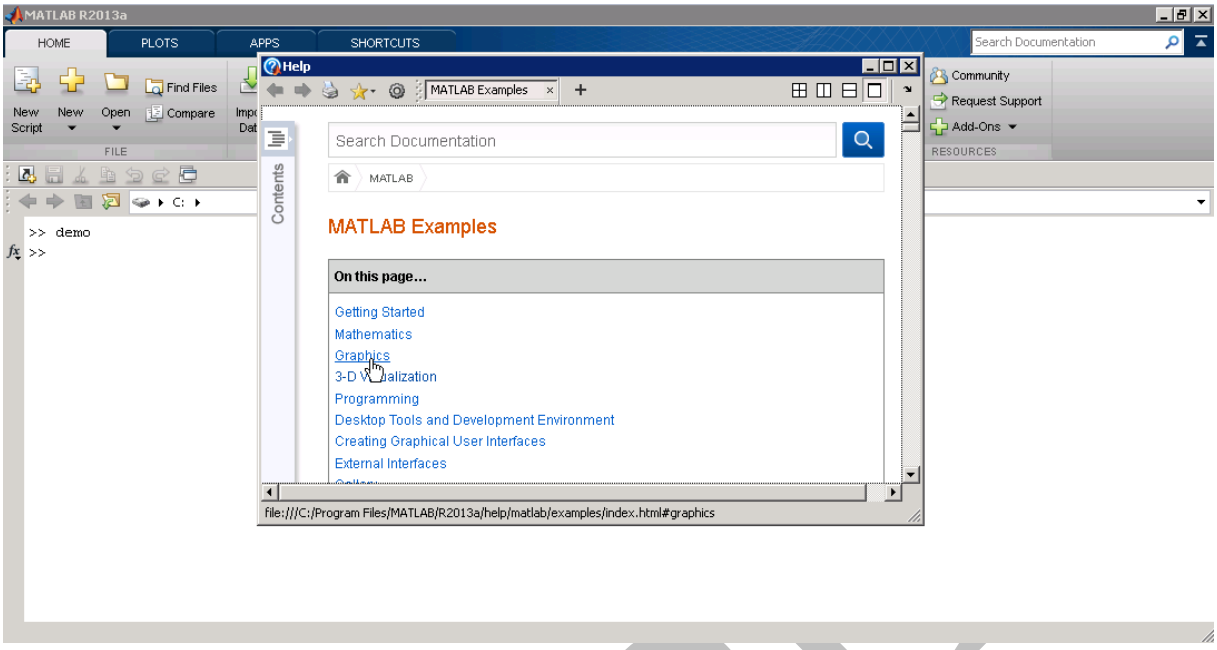


Or **graf3d**:

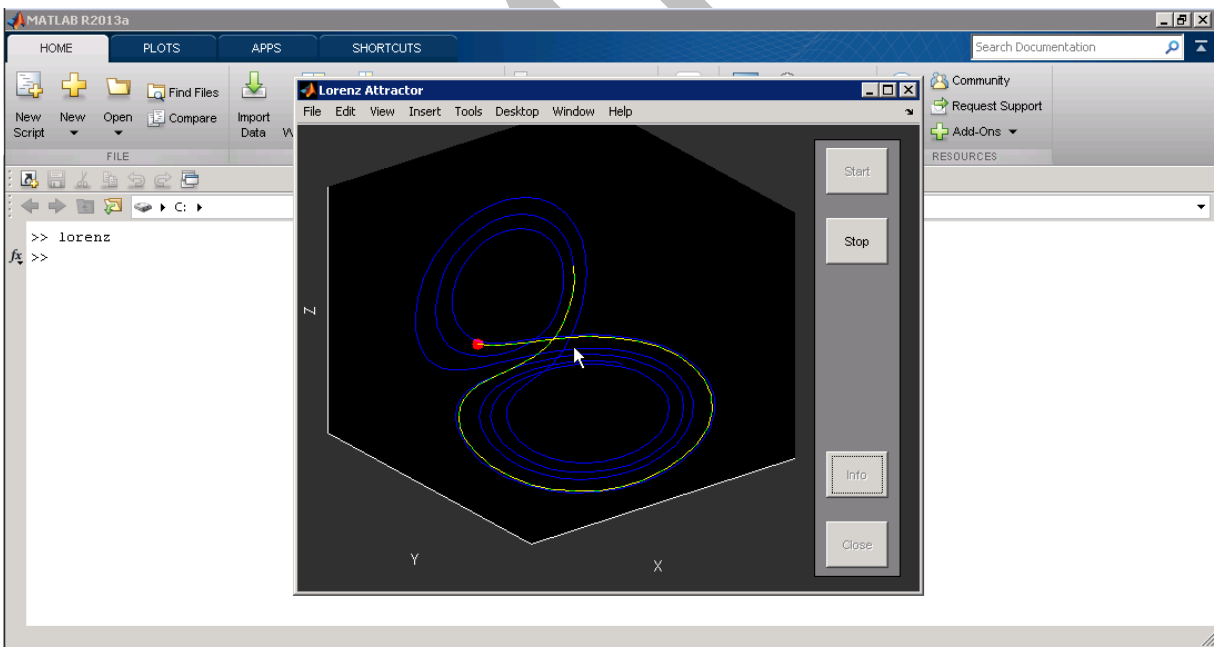


Each of these charts can be edit and you have access to the source code to see how they were builded.

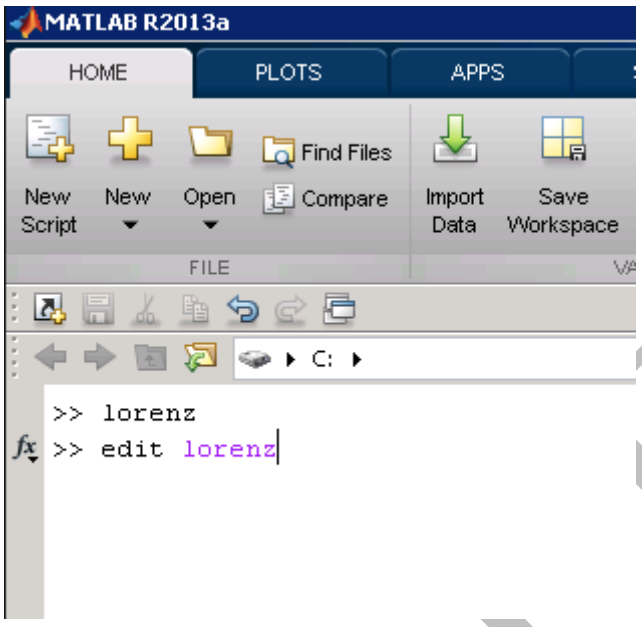
If you type the command **demo** you will see a section *Graphics*:



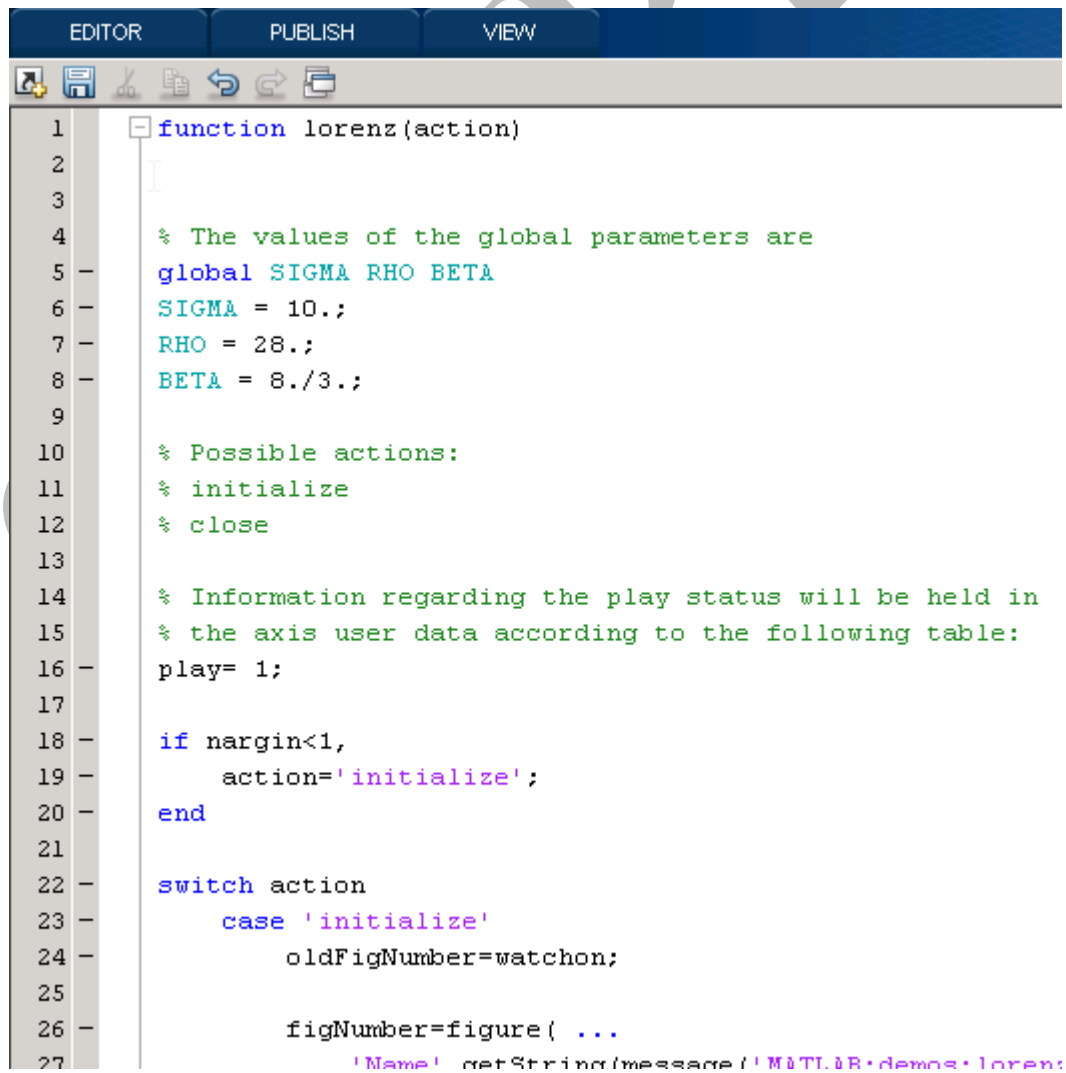
with a lot of possible demo charts commands. For example the famous Lorenz attractor using the command **lorenz**:



And as says before you can see the code source by using the well know command **edit**:

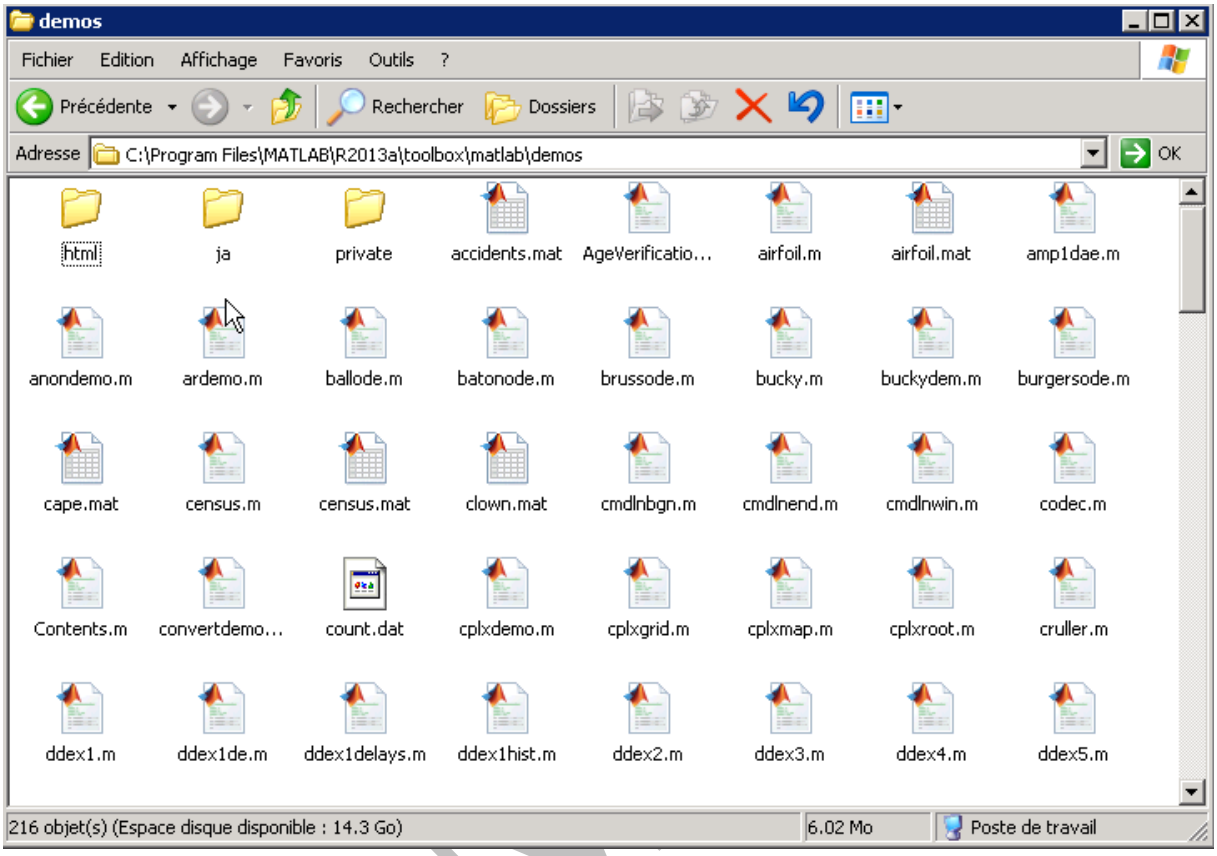


Thus in this particular example you will get:

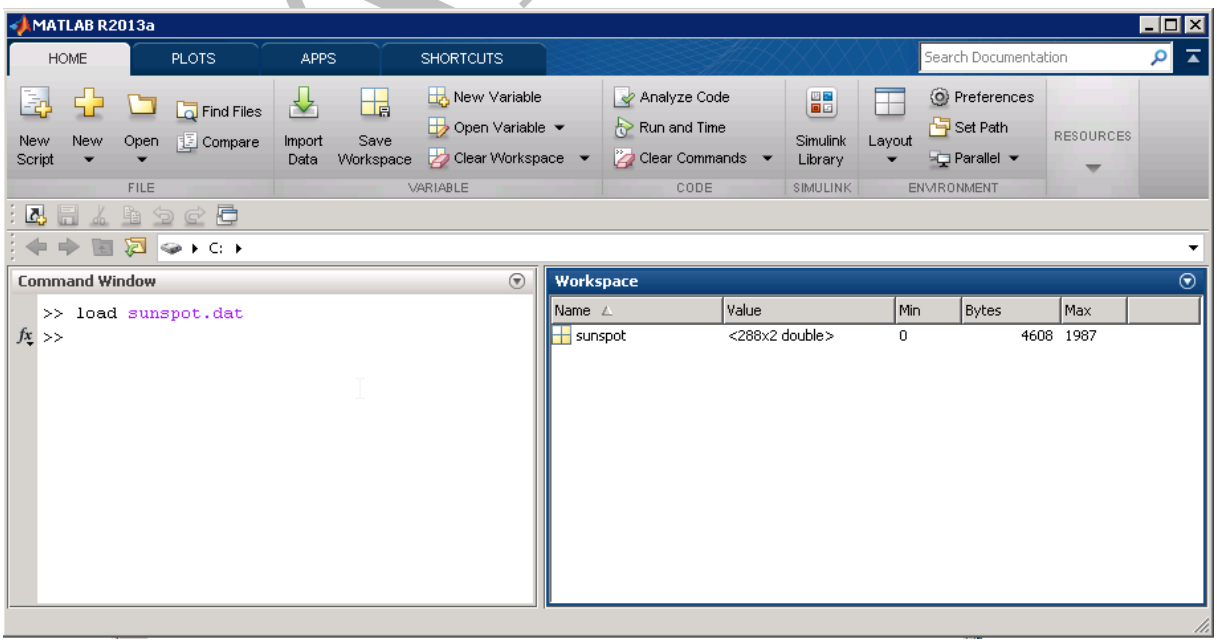


Data sets

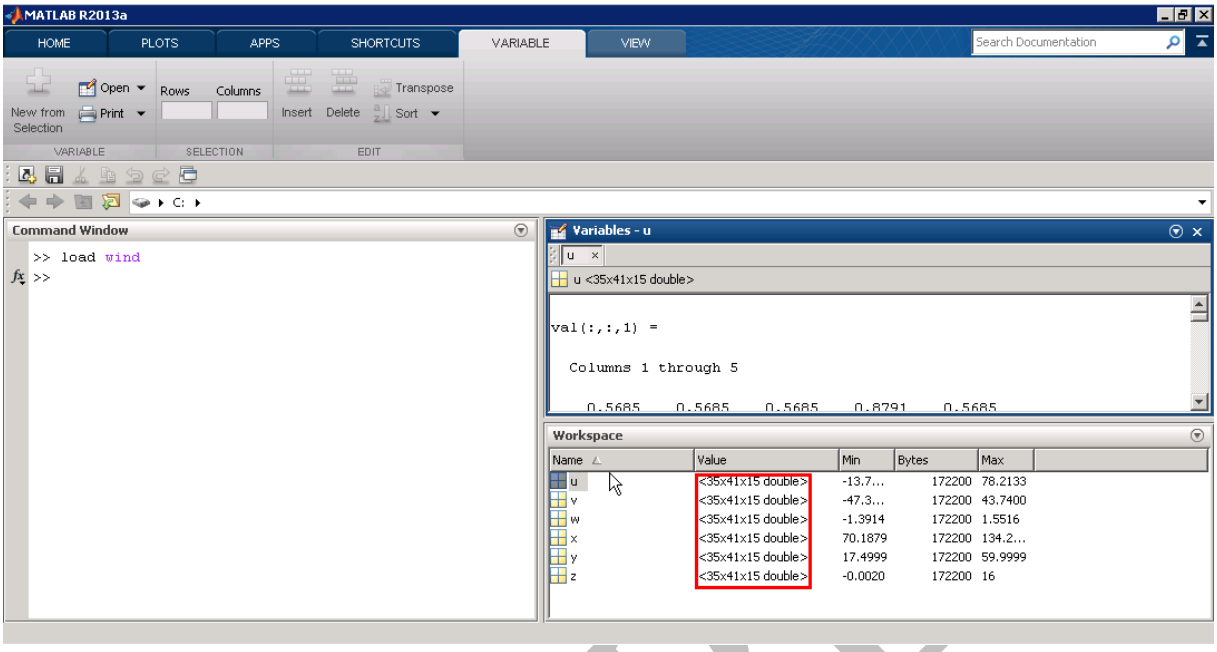
Otherwise if you want to play to create your own charts but you do not have any datasets to play with, MATLAB provides a few datasets in the folder *demos* of your hard drive:



For example the small *sunspot* data set can be loaded with the simple well know command **load**:



Or for a more complicated example you can use the *wind* data set (we will use it a lot later in this book):



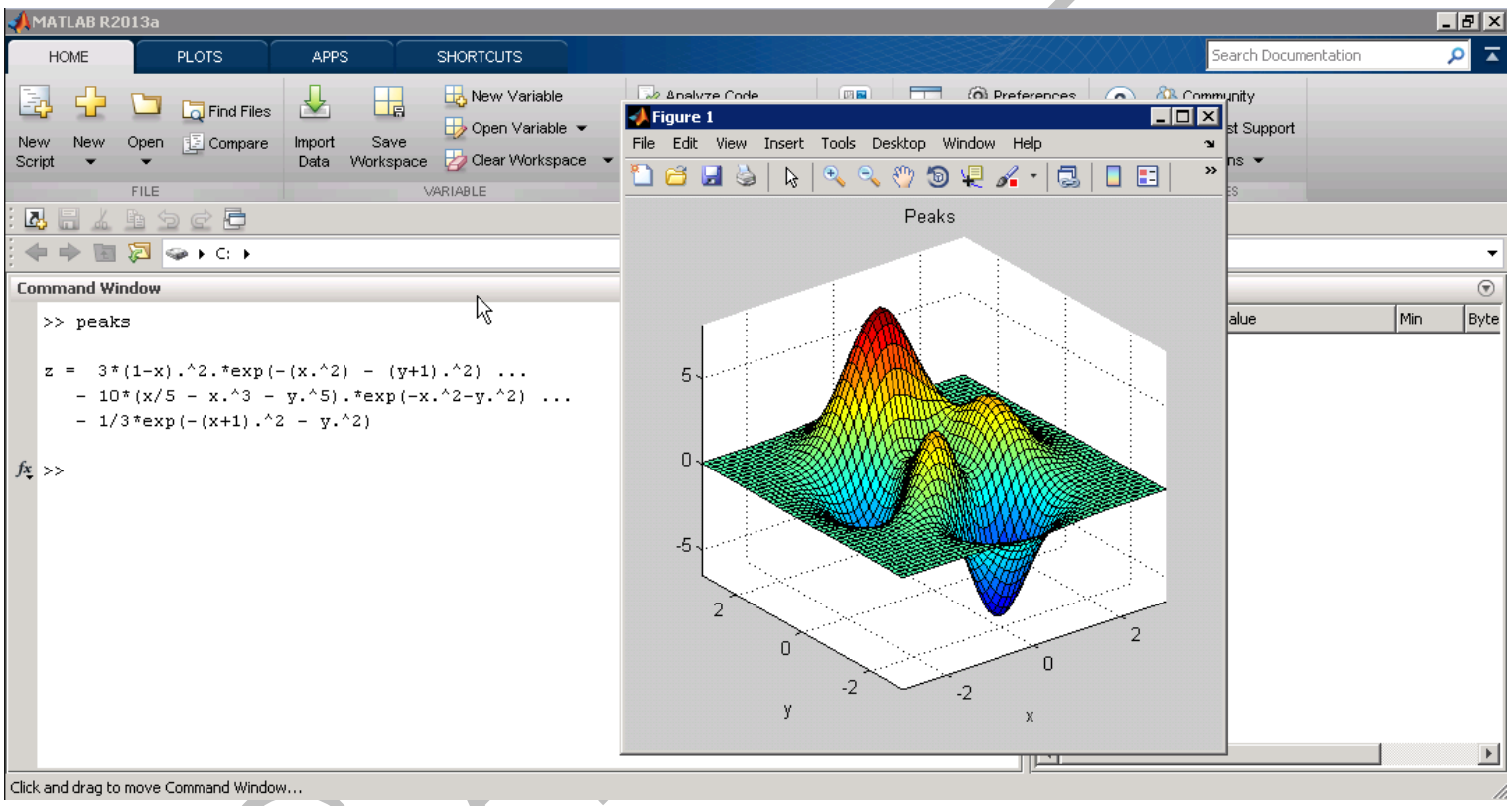
SAMPLE

Chart wizard

For users that don't like to write commands or script to create charts there is a chart wizard in MATLAB but for your information... you will be quickly limited if you learn only this way of doing charts.

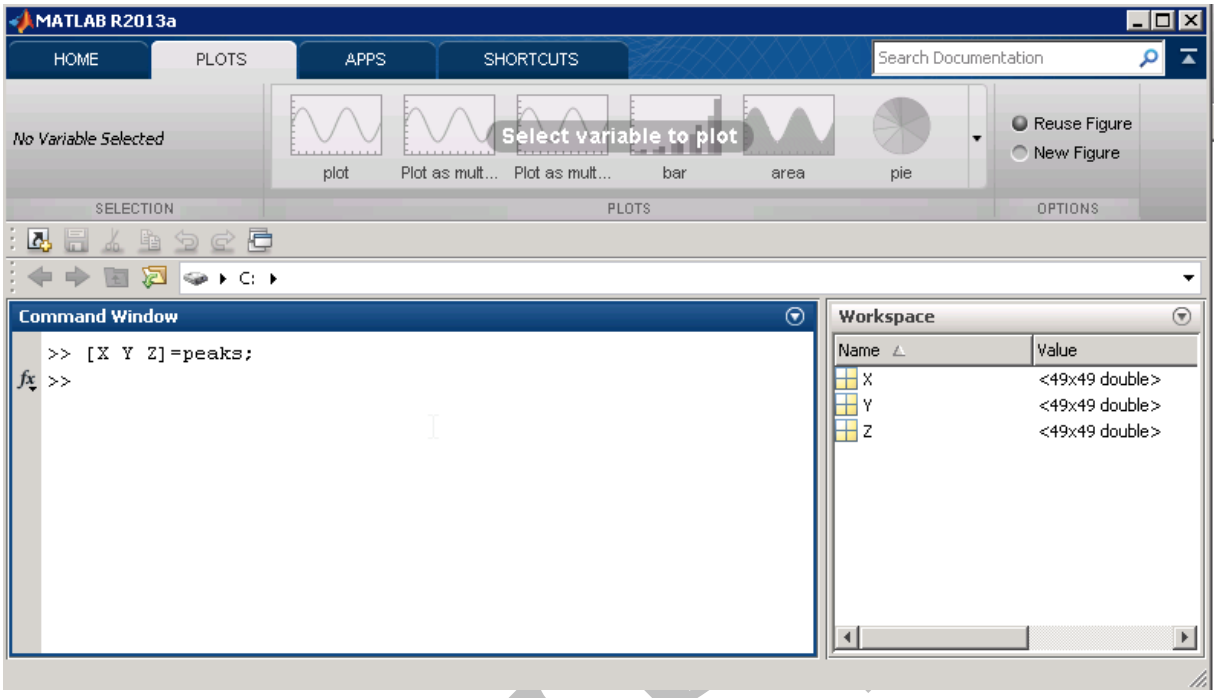
But this wizard is still sometimes useful to remember a simple code or command that we have forgotten and when we don't have an Interne access in our company to search for some help or documentation on Google. This is way it is still useful to have a quick look at it.

For this purpose we will play with the old famous command **peaks**:

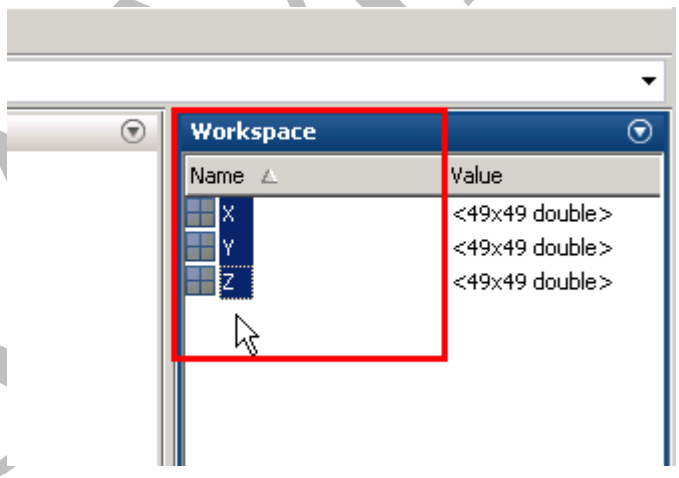


OK let us start!

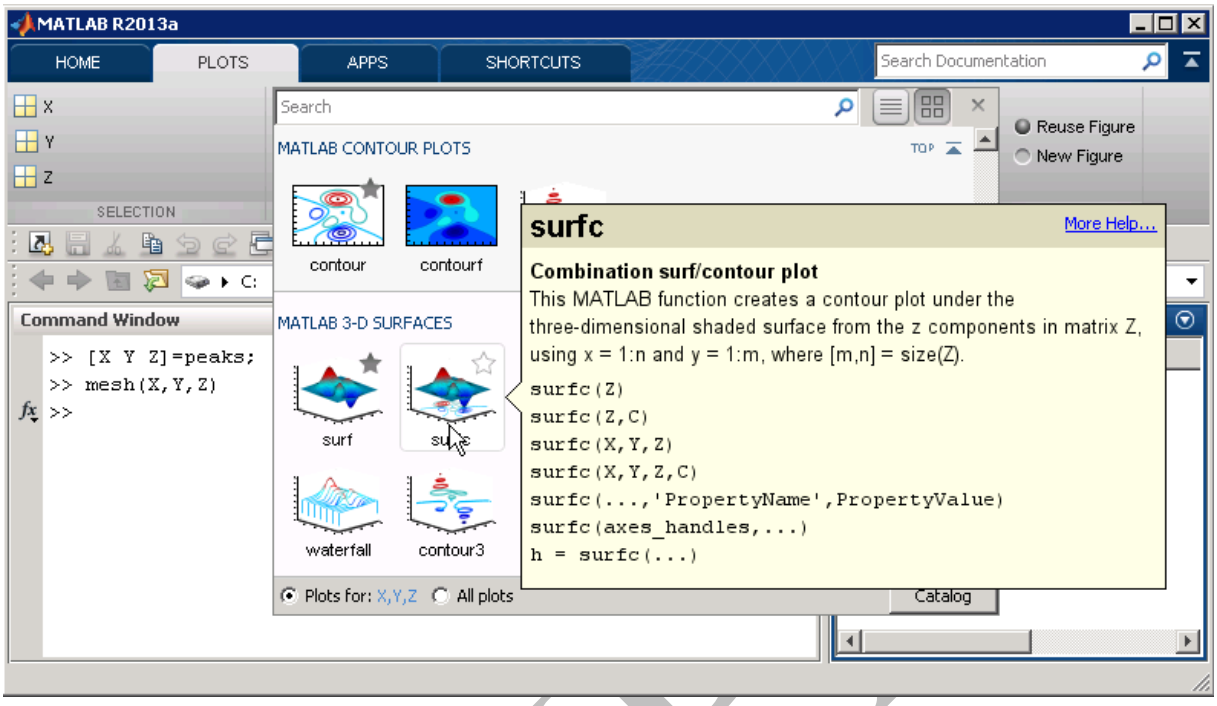
First we import separately the matrices of the peak function as following:



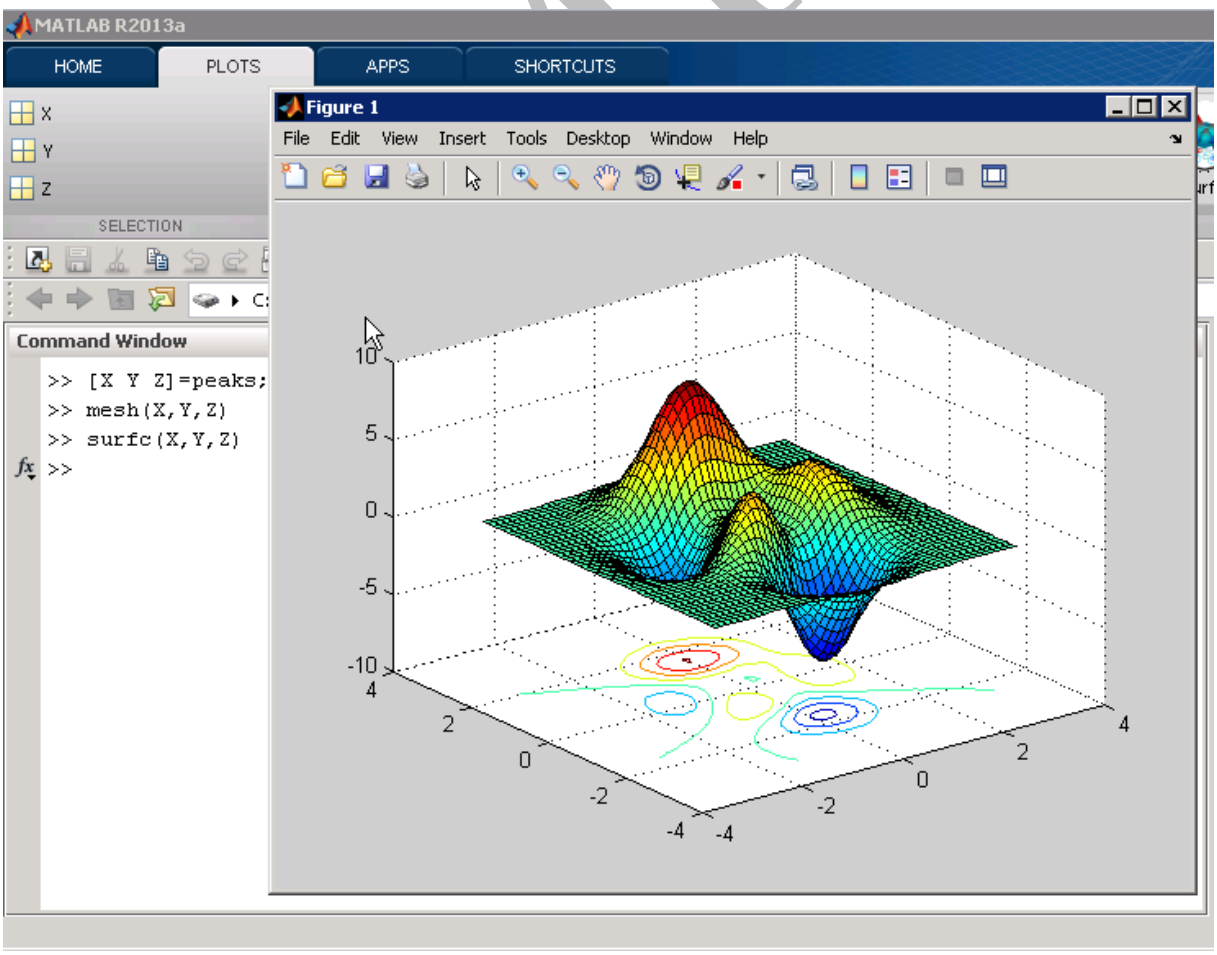
As you can see we have three matrices (this is commonly named a "mesh grid" in the MATLAB world):



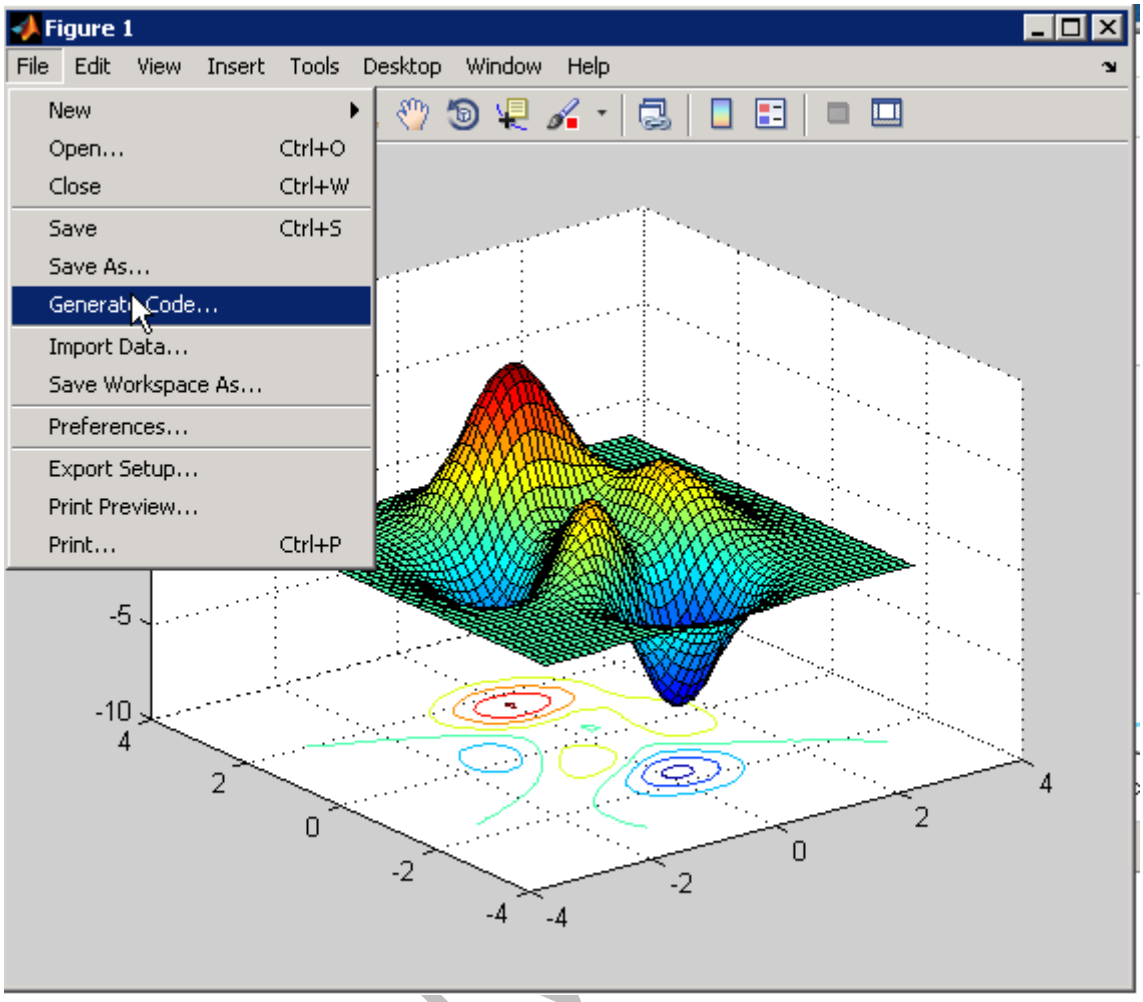
Once this done you go in the ribbon **PLOTS** and select for example the **surfc** MATLAB 3-D surfaces:



You will get the following result:



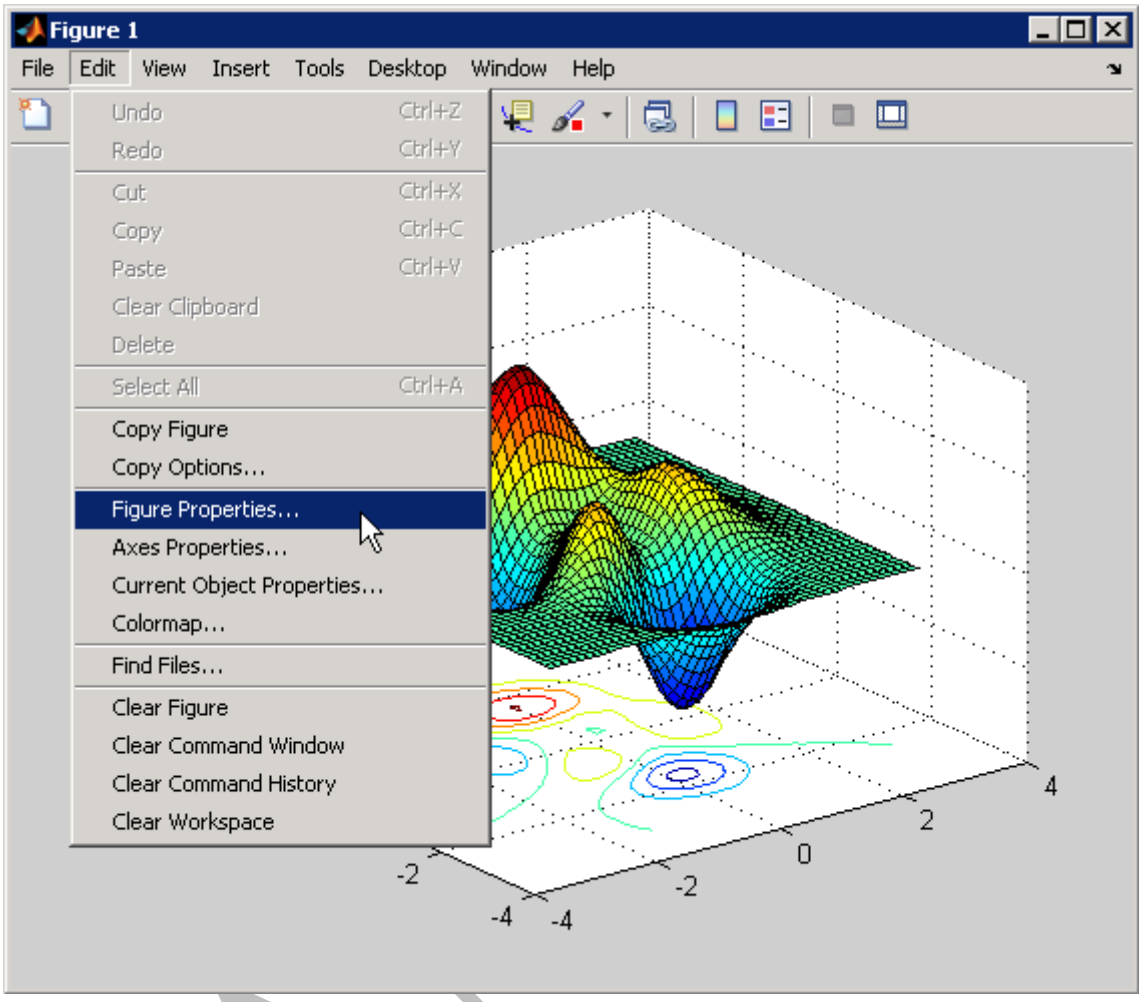
Now if you go in the menu **File/Generate Code...**:



MATLAB will give you automatically all the (awful) equivalent script that can reproduce this chart later automatically (I did not make a screenshot of this code because is very awful).

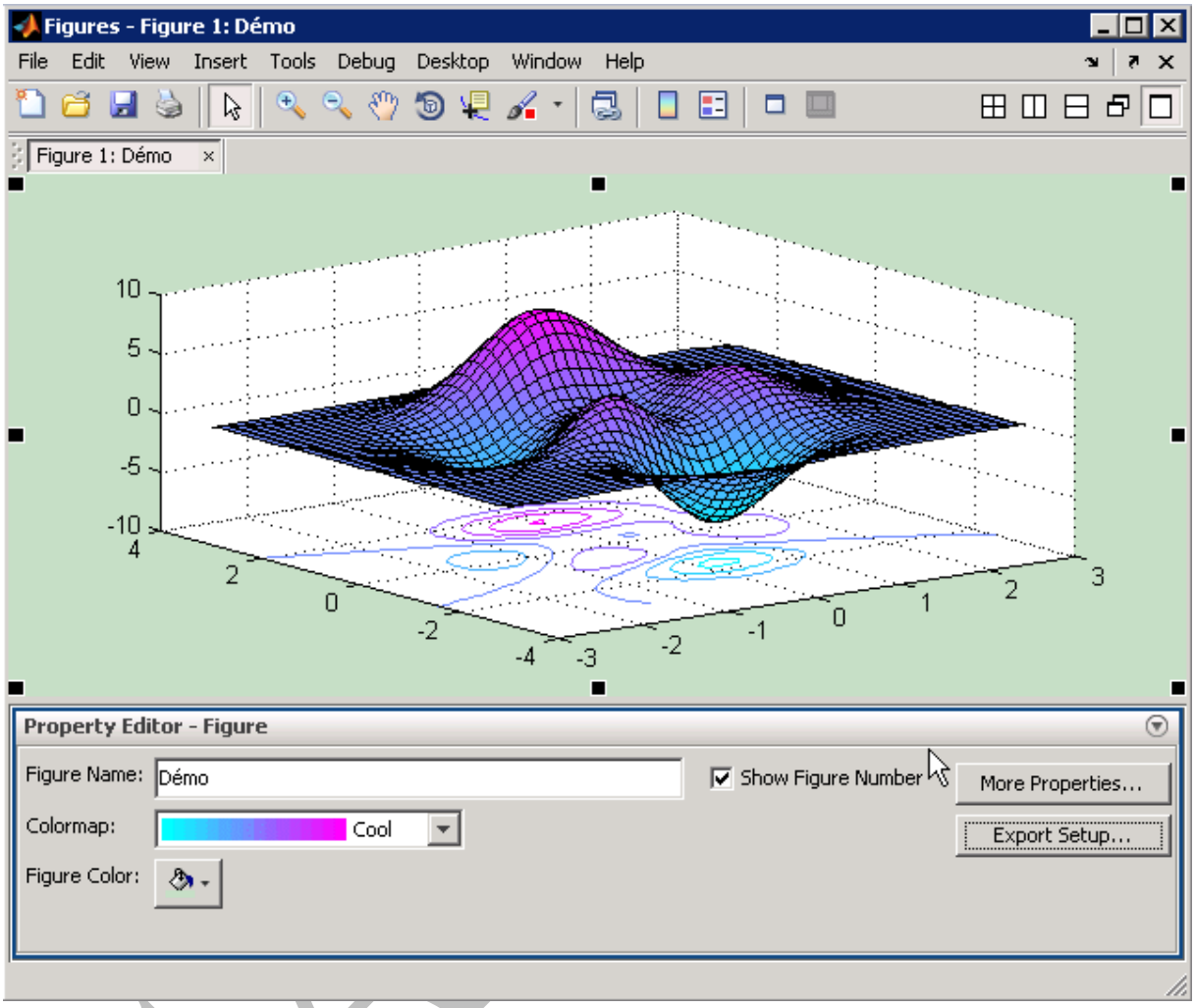
SAMPLE

Now to customize any chart, when you have the menu visible (because it can be hidden), one possibility is to go on the menu **Edit/Figure Properties...**:

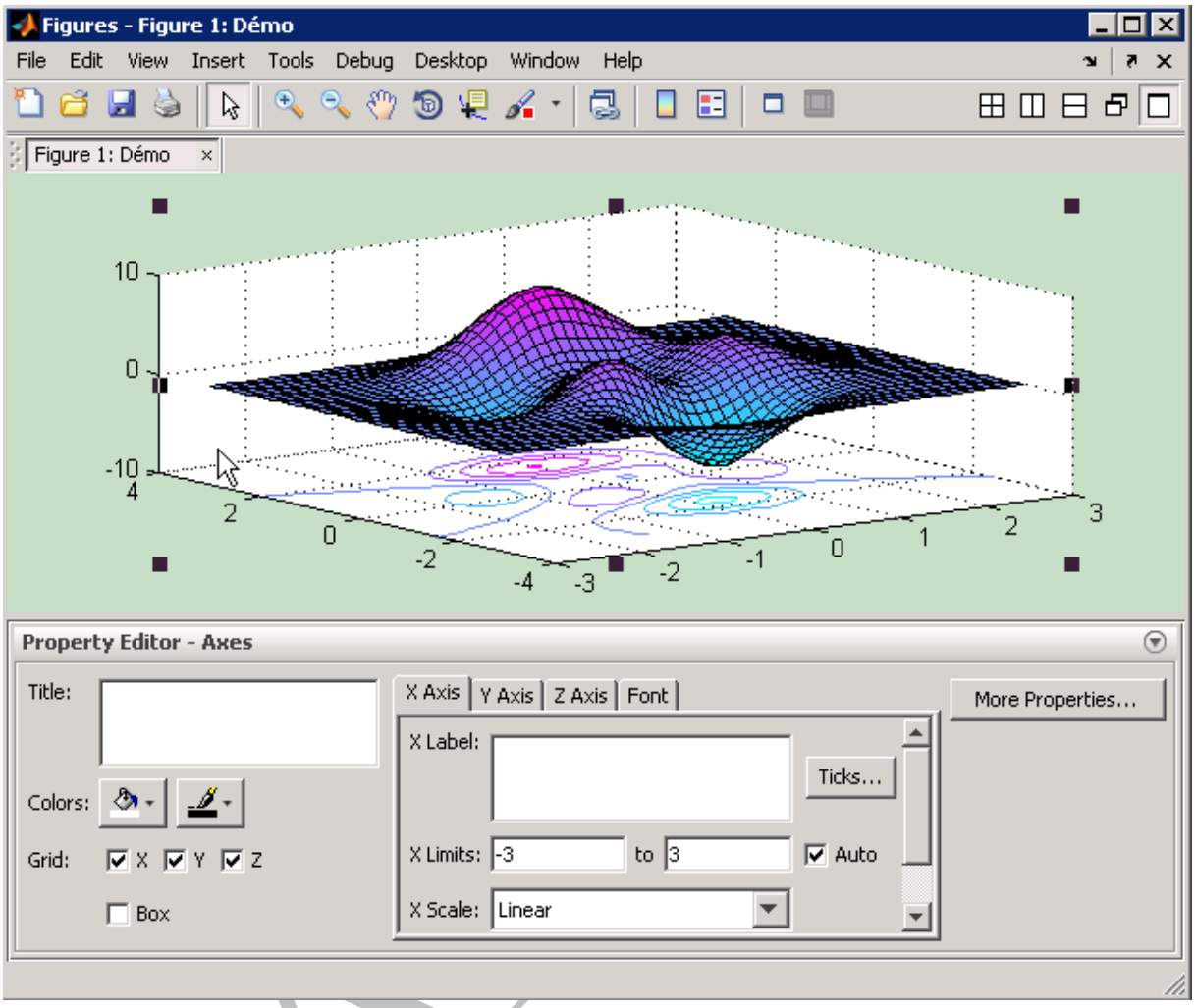


SAMPLE

Then the chart window will change a little bit and you will have hundreds of option to customize the chart as for example (see below) the background color of the chart or its colormap and also if you want the figure number to be visible or not in the tible bar of the windows, etc.:

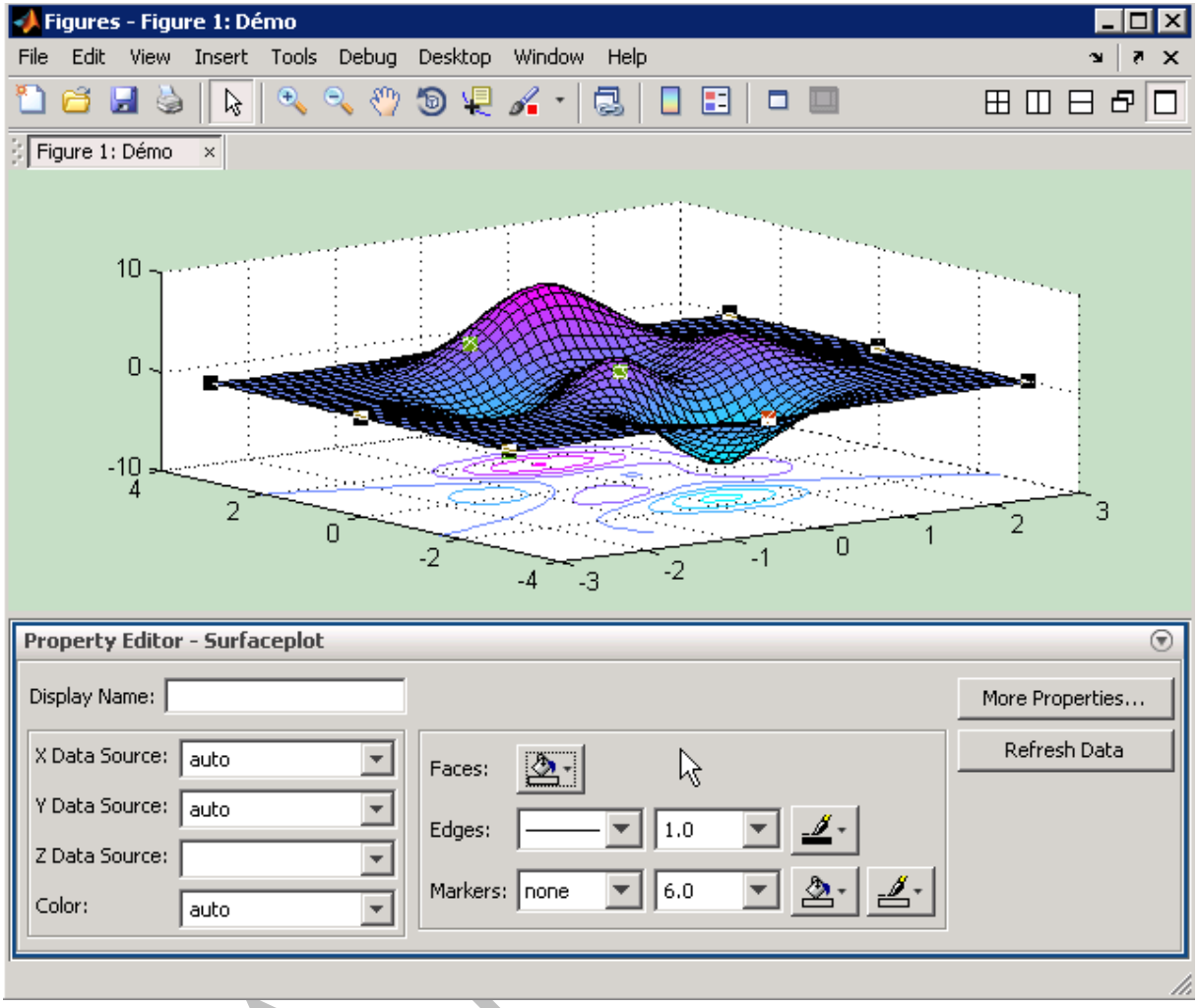


If you double click on the axis, the bottom part will change with other customization options that are sometimes very useful and important (for the people that don't like to script everything themselves!):



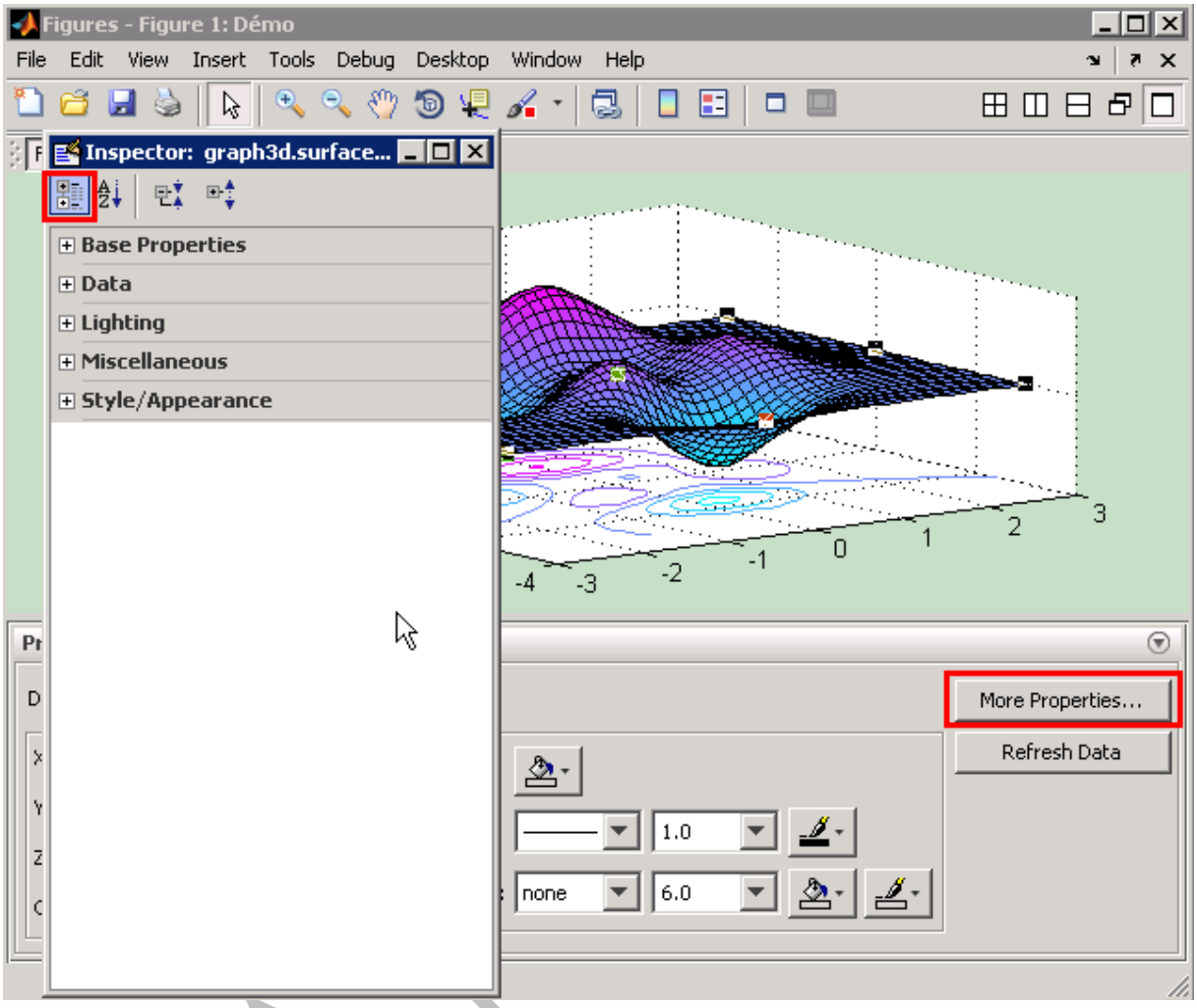
SA

Or if you double click on the surface you can also change a few options:



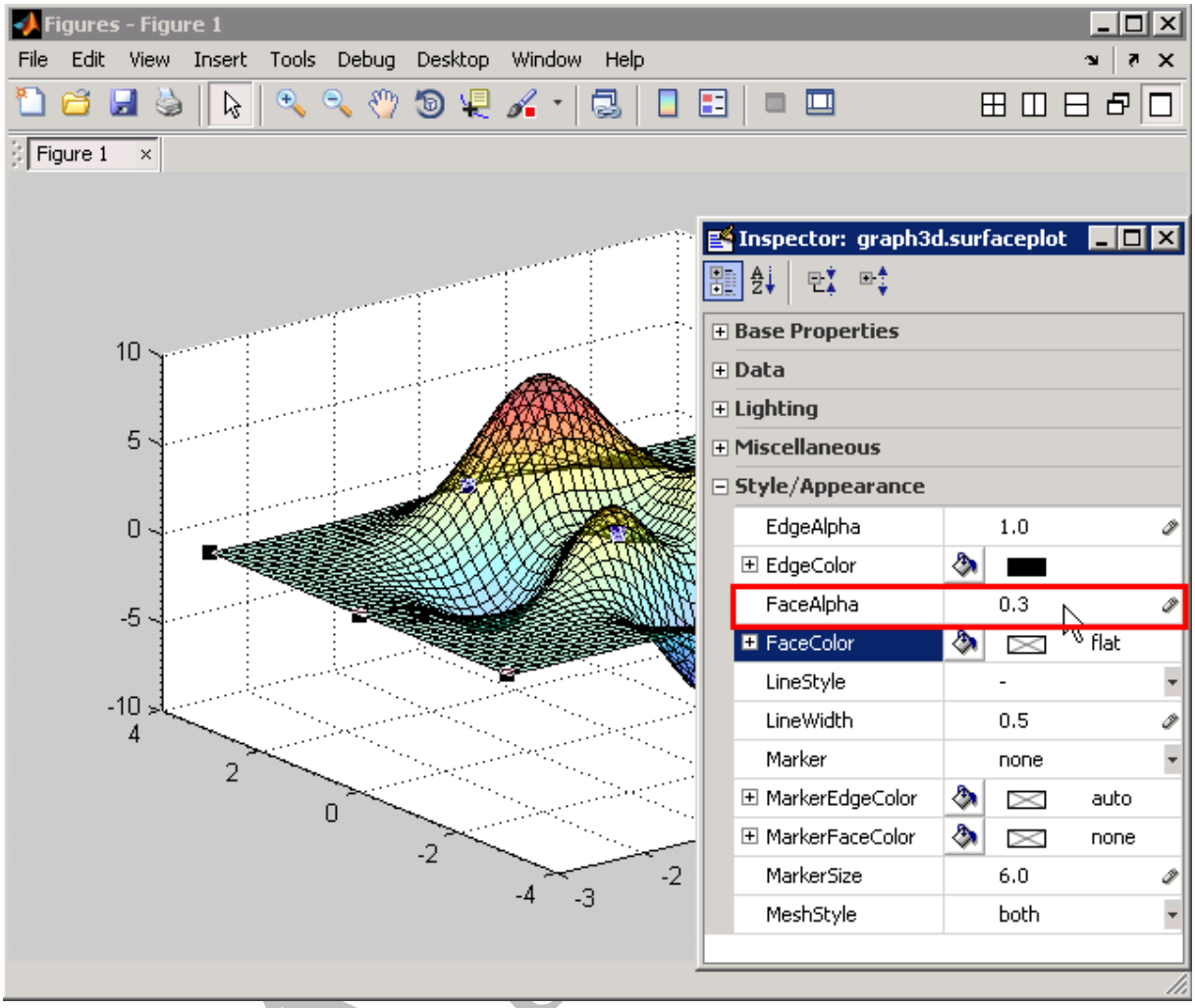
SAMPLE

If you click on the button **More Properties...** you will see a small **Inspector** window:



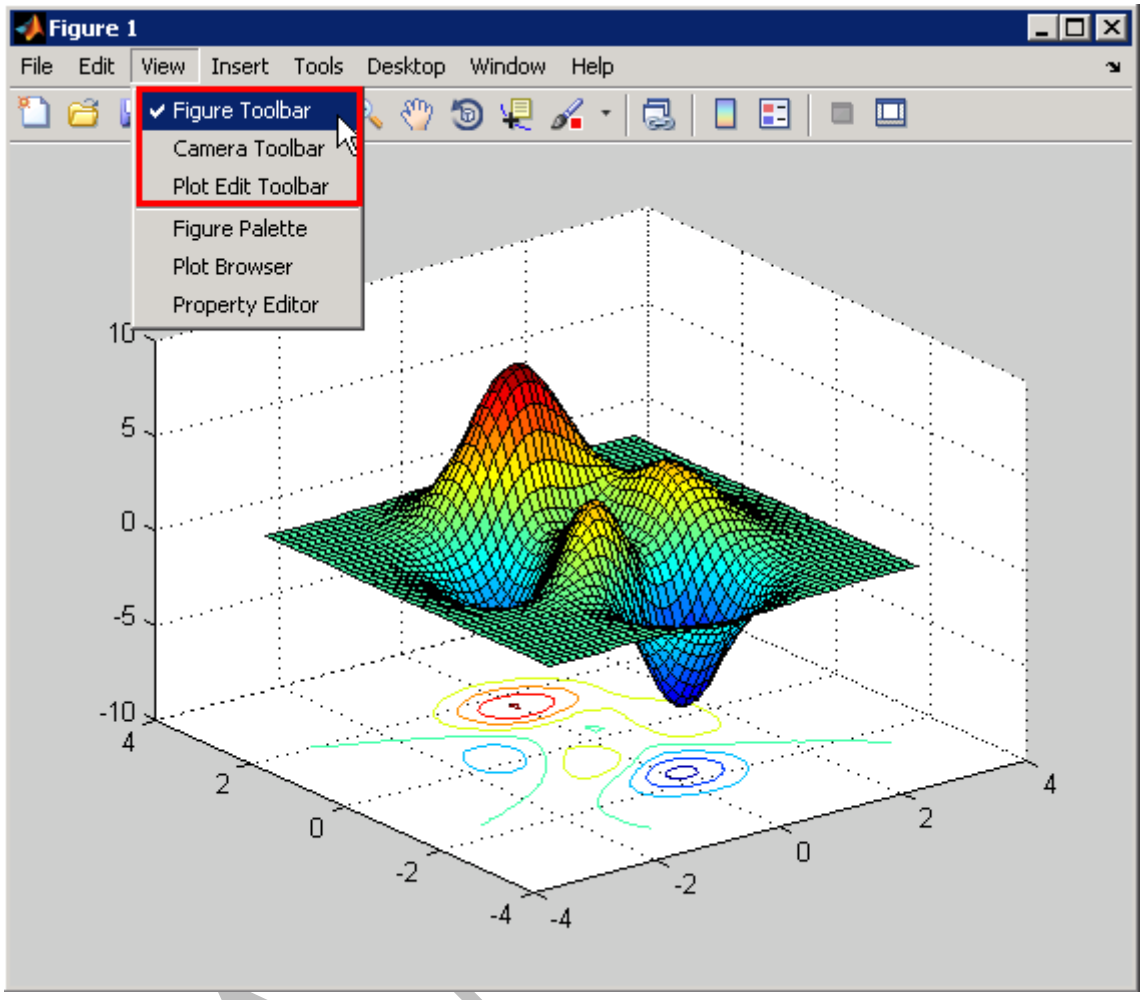
SAMPLE

Where you can change almost a hundred of properties for your charts (for example we changed here the transparency of the surface by playing the **FaceAlpha** property):



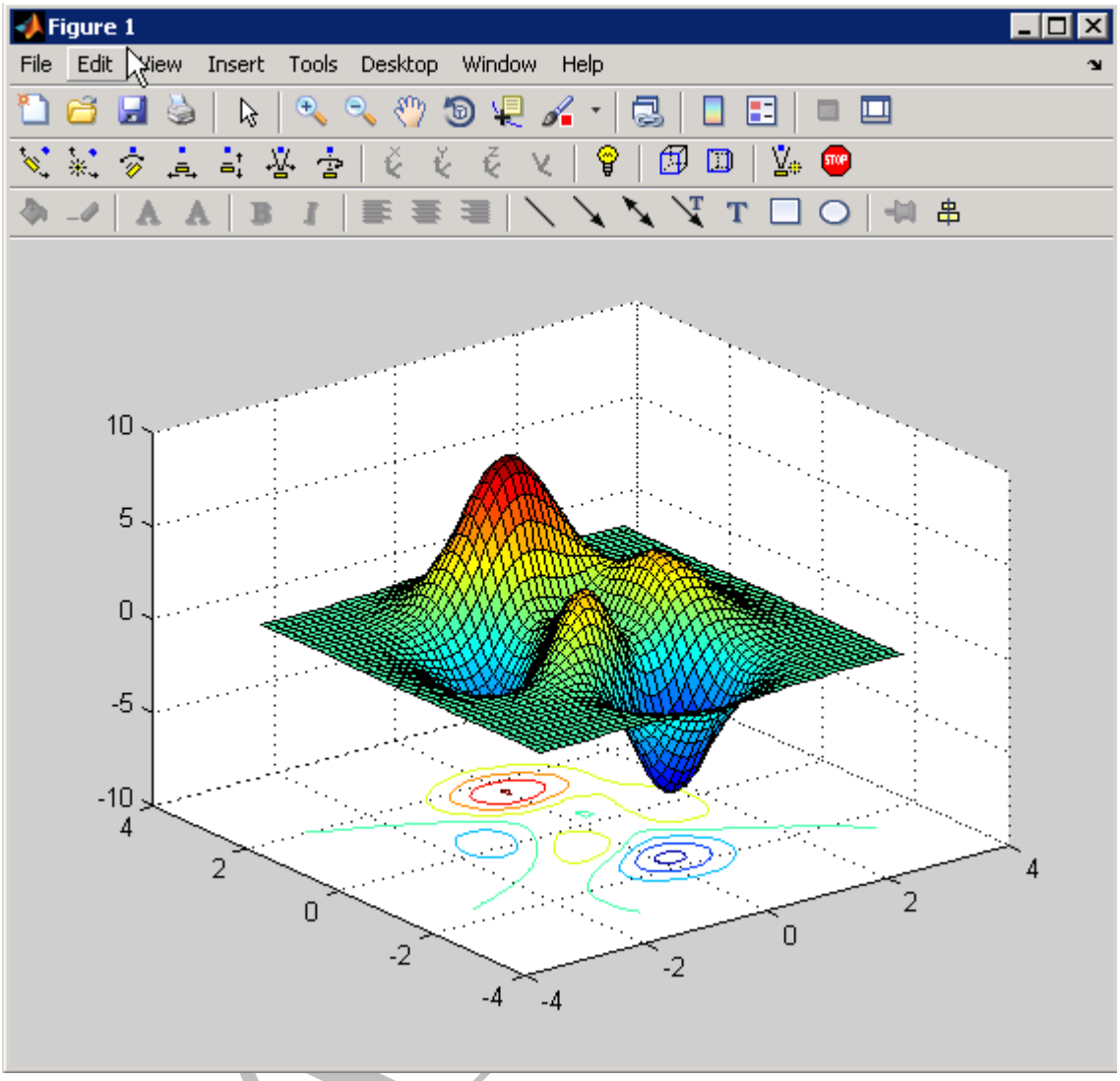
SA

Ok, now see something else a structural or cosmetic properties. Have a look at the available toolbars:

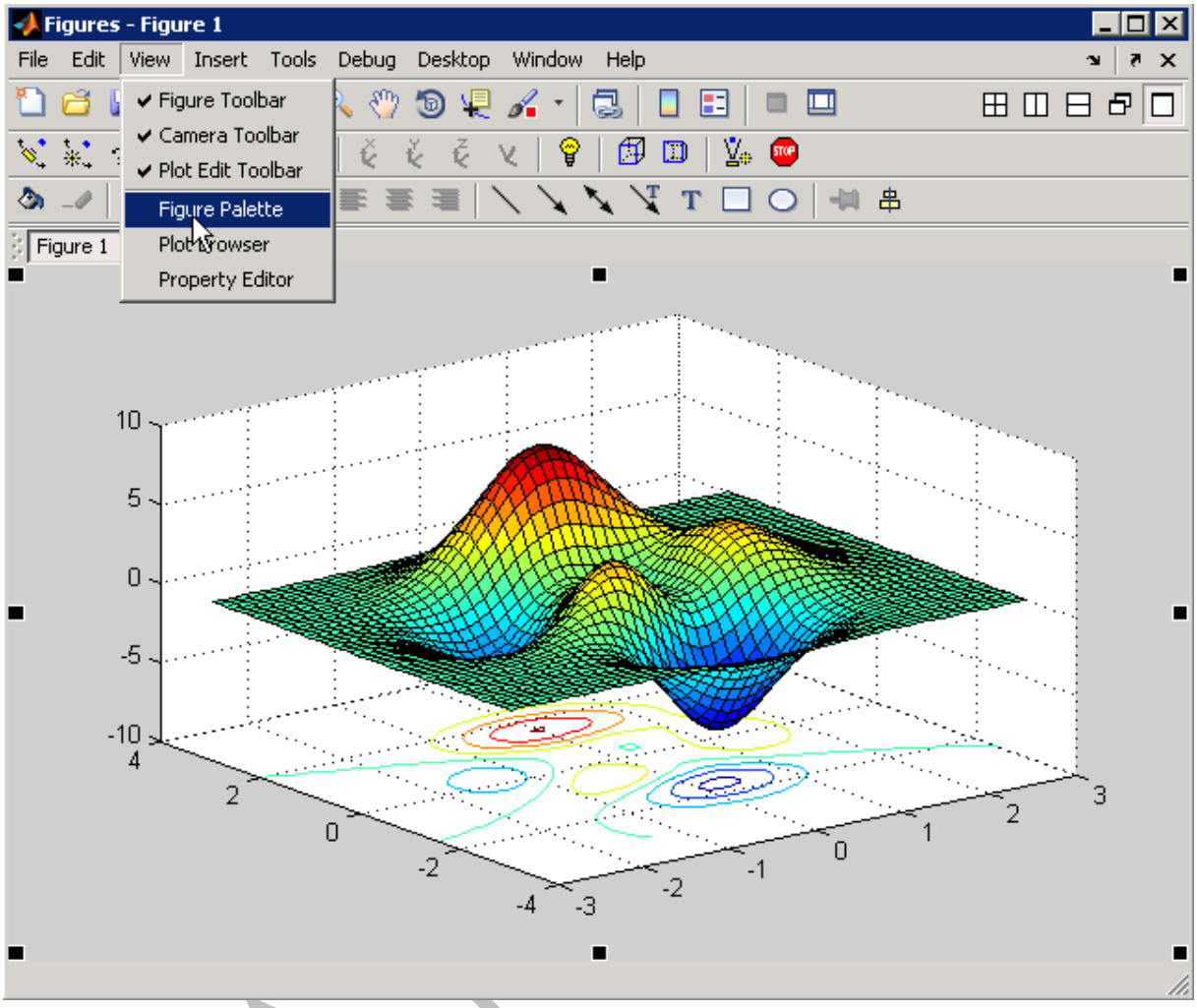


SAM

If you activate all the toolbars the result will looks like following:

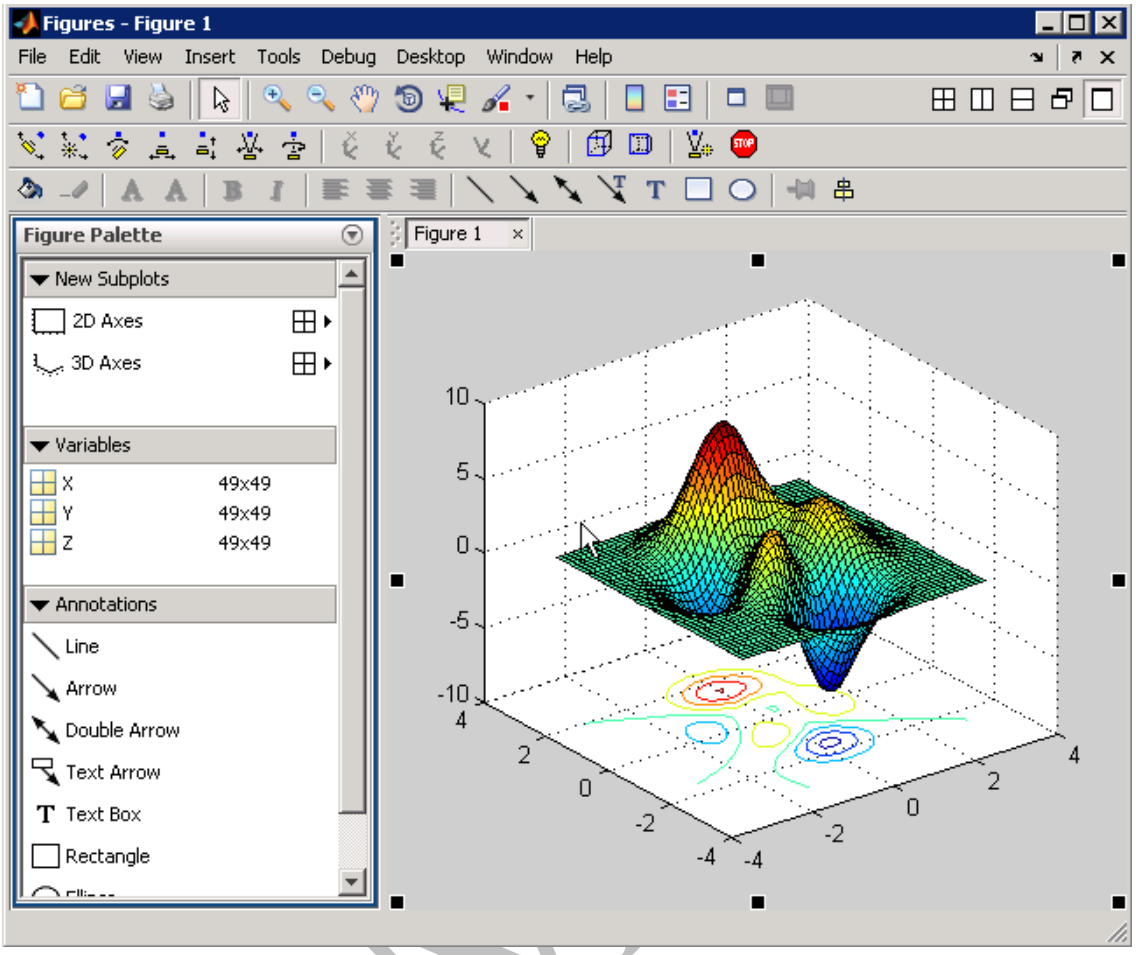


There is also a **Figure Palette** especially useful to create subplots with the wizard:

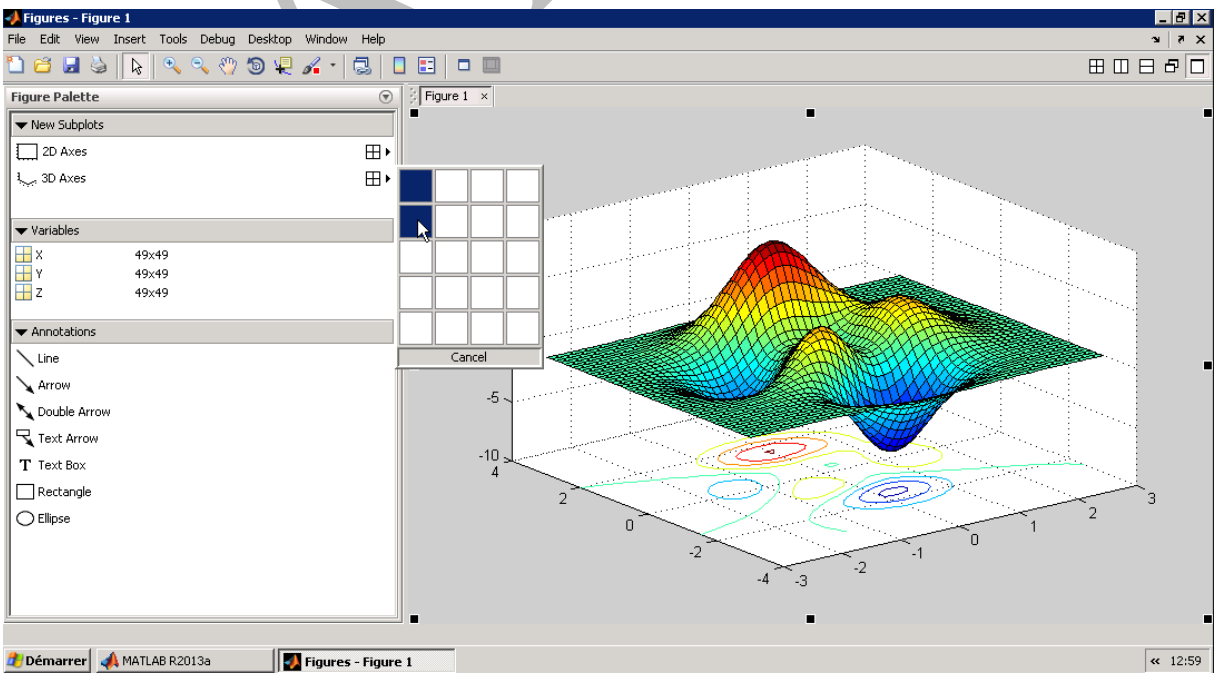


SAM

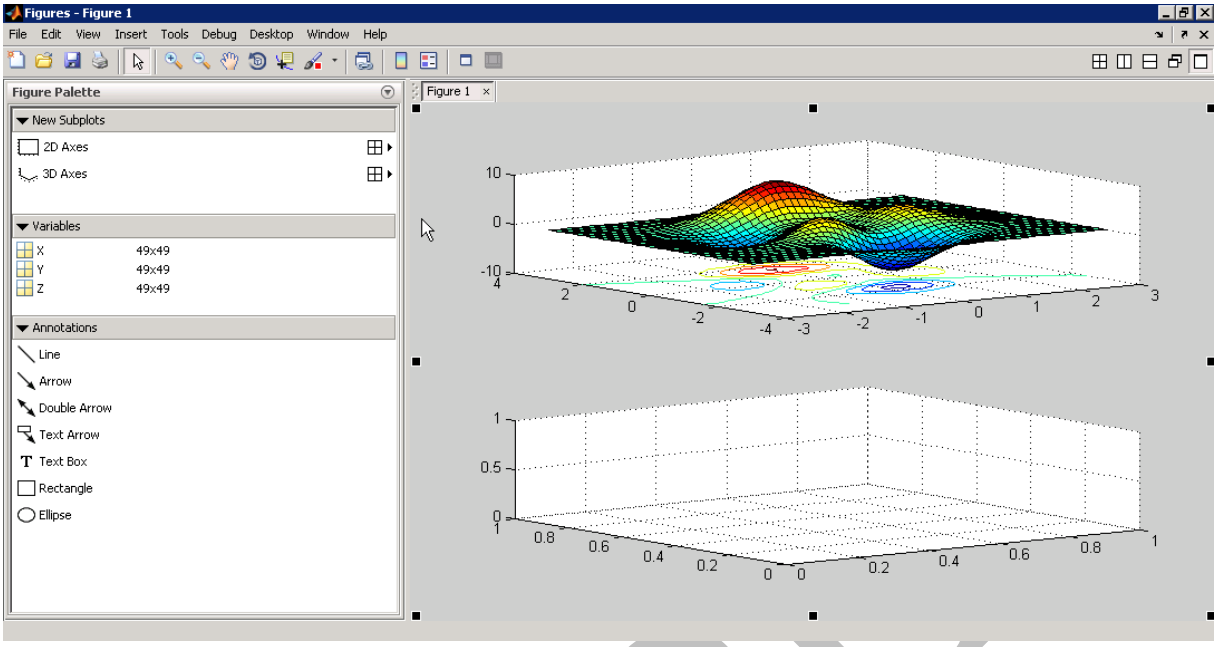
The **Figure Palette** looks like this:



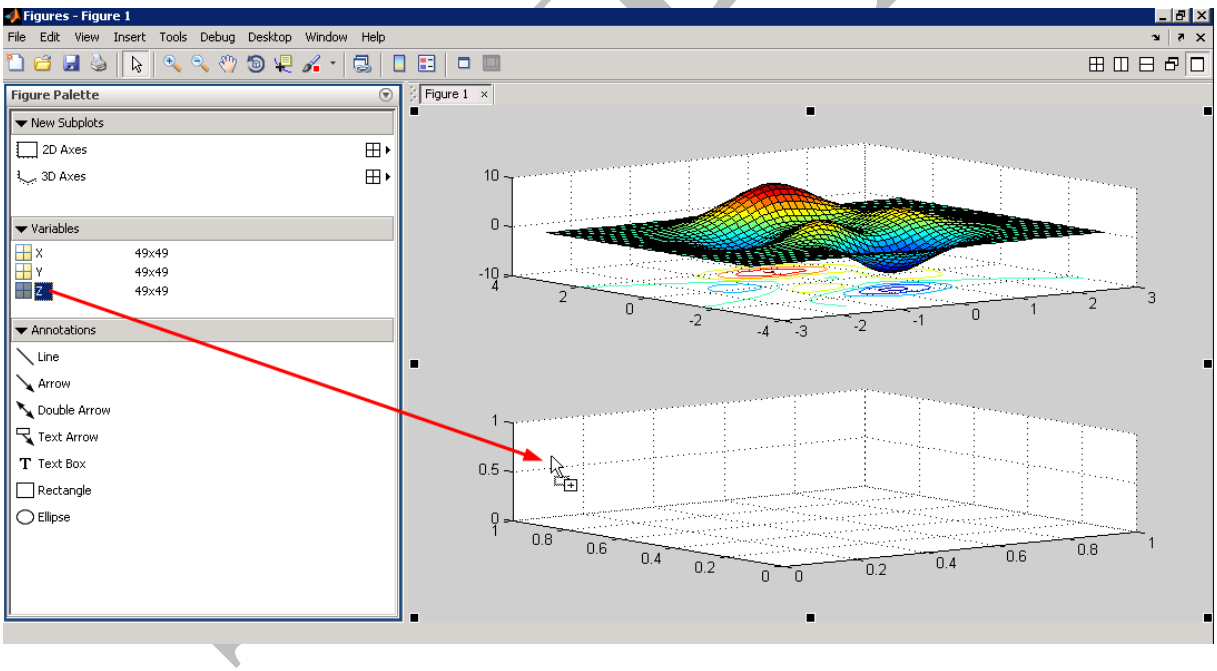
If you click on 3D axes arrows of the **Figure Palette** panel you will be able to create a few subplots because (you can also mix 3D and 2D subplots in the same figure!):



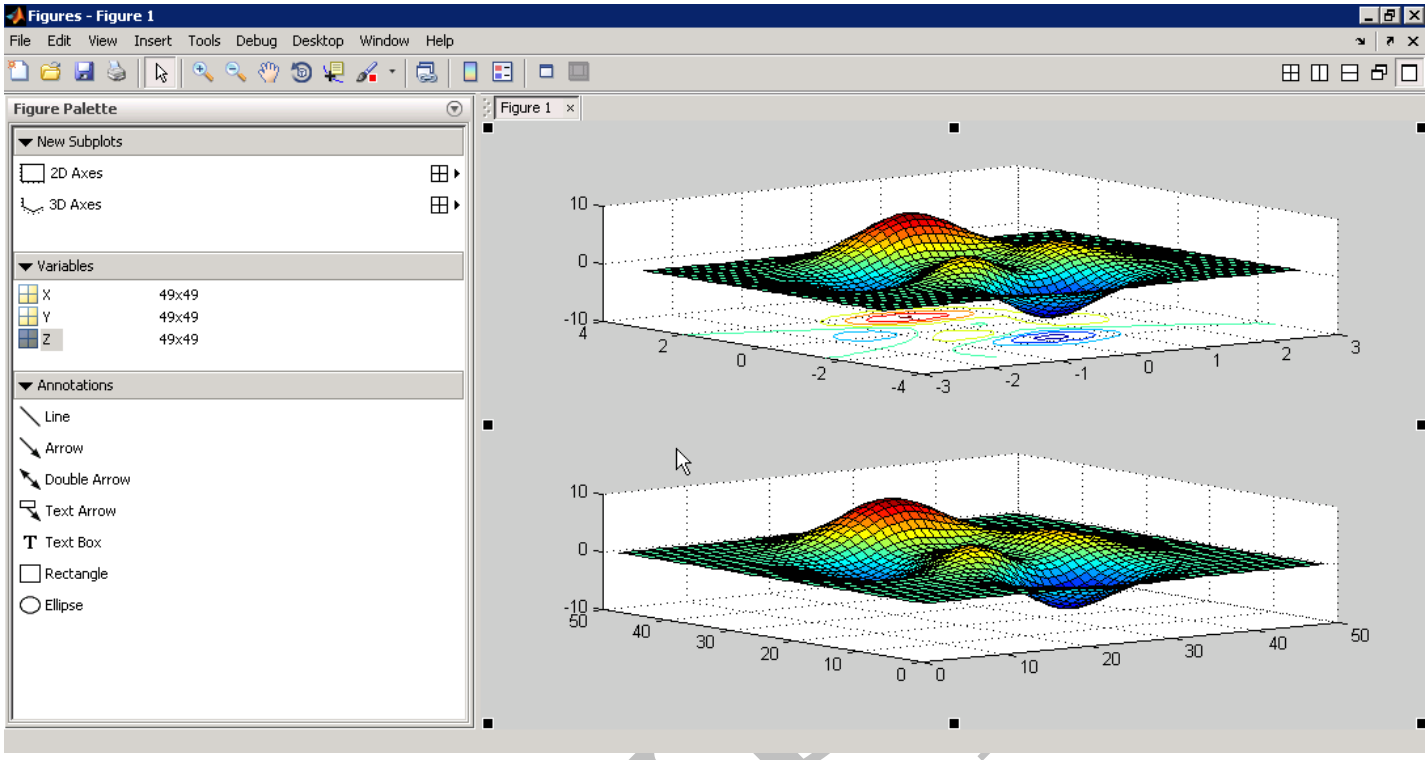
For example with an additional toolbox the result looks like this:



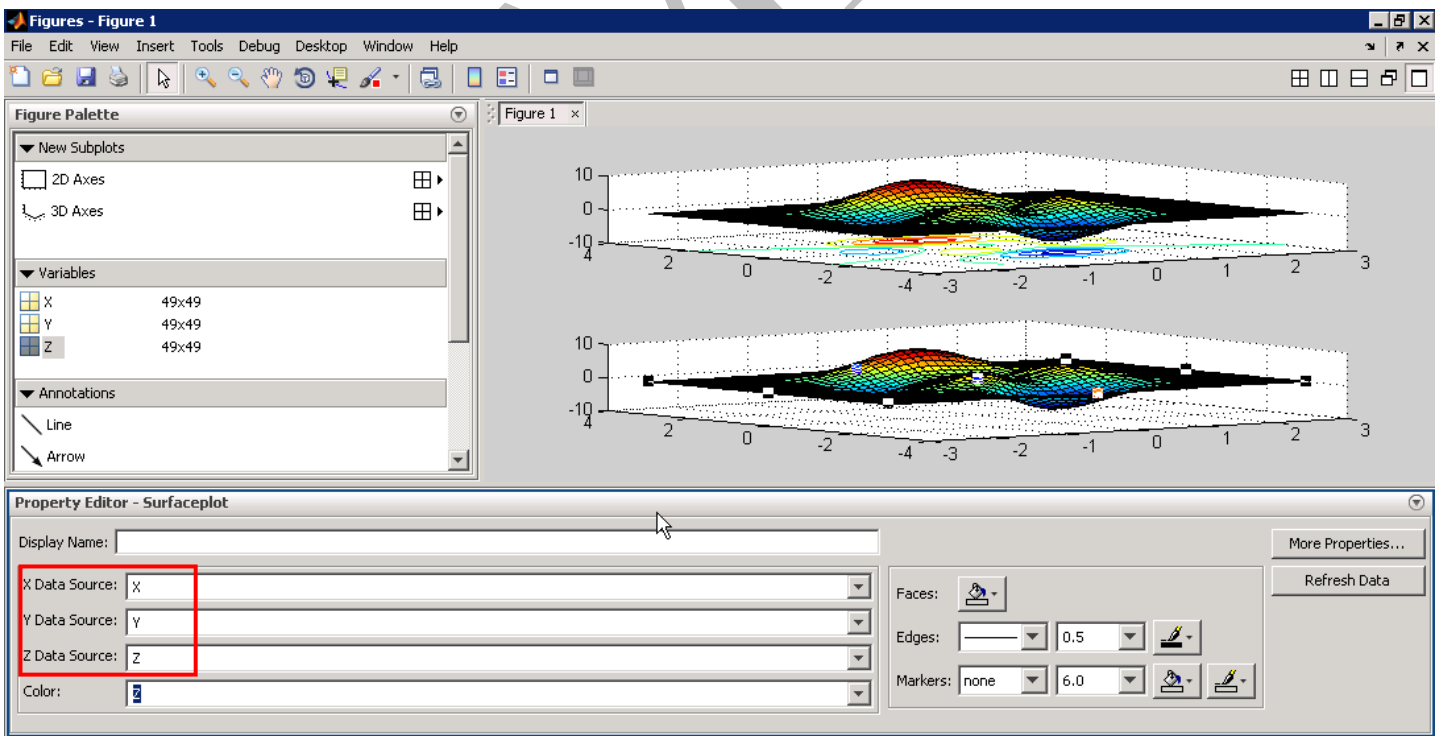
Then you just have to drag the components that you want to plot in this new 3D or 2D suplot:



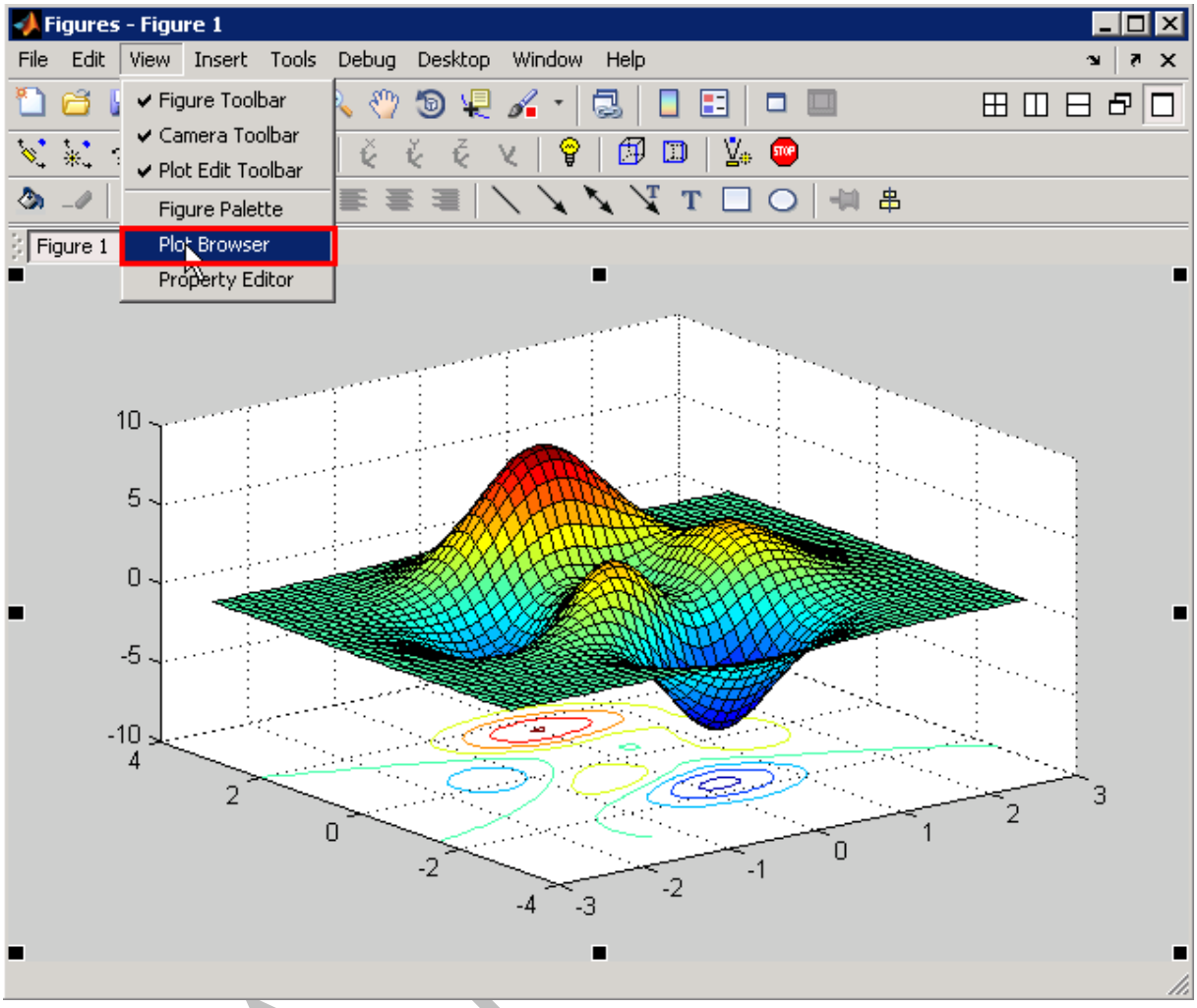
Here for example we just reproduced the above one (for sure this example is almost stupid...):



Now if you double click on the surface you can always change after the source values for the second plot to have the correct X, Y and Z components as you can see below:

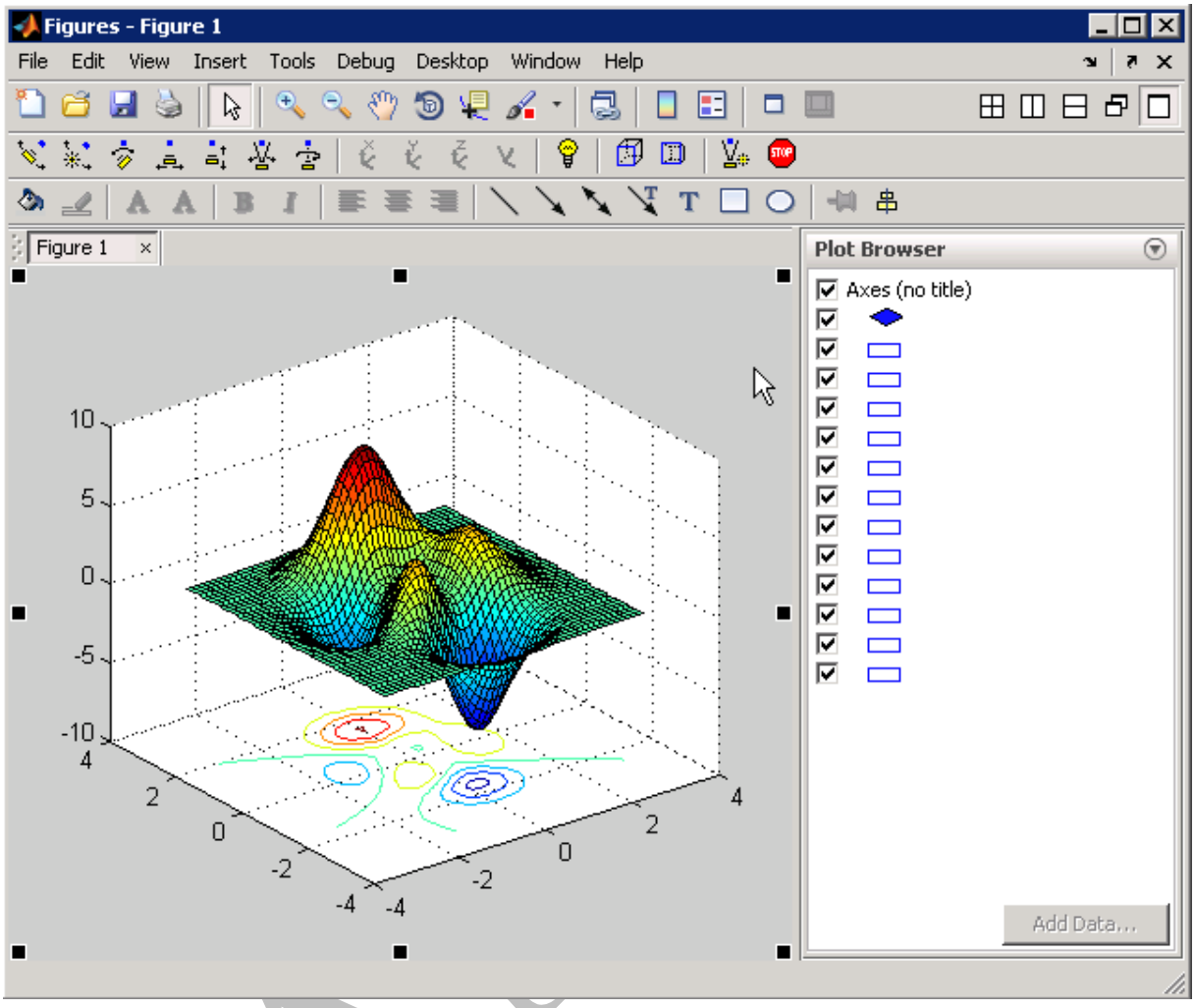


Now take a look the **Plot Browser**:



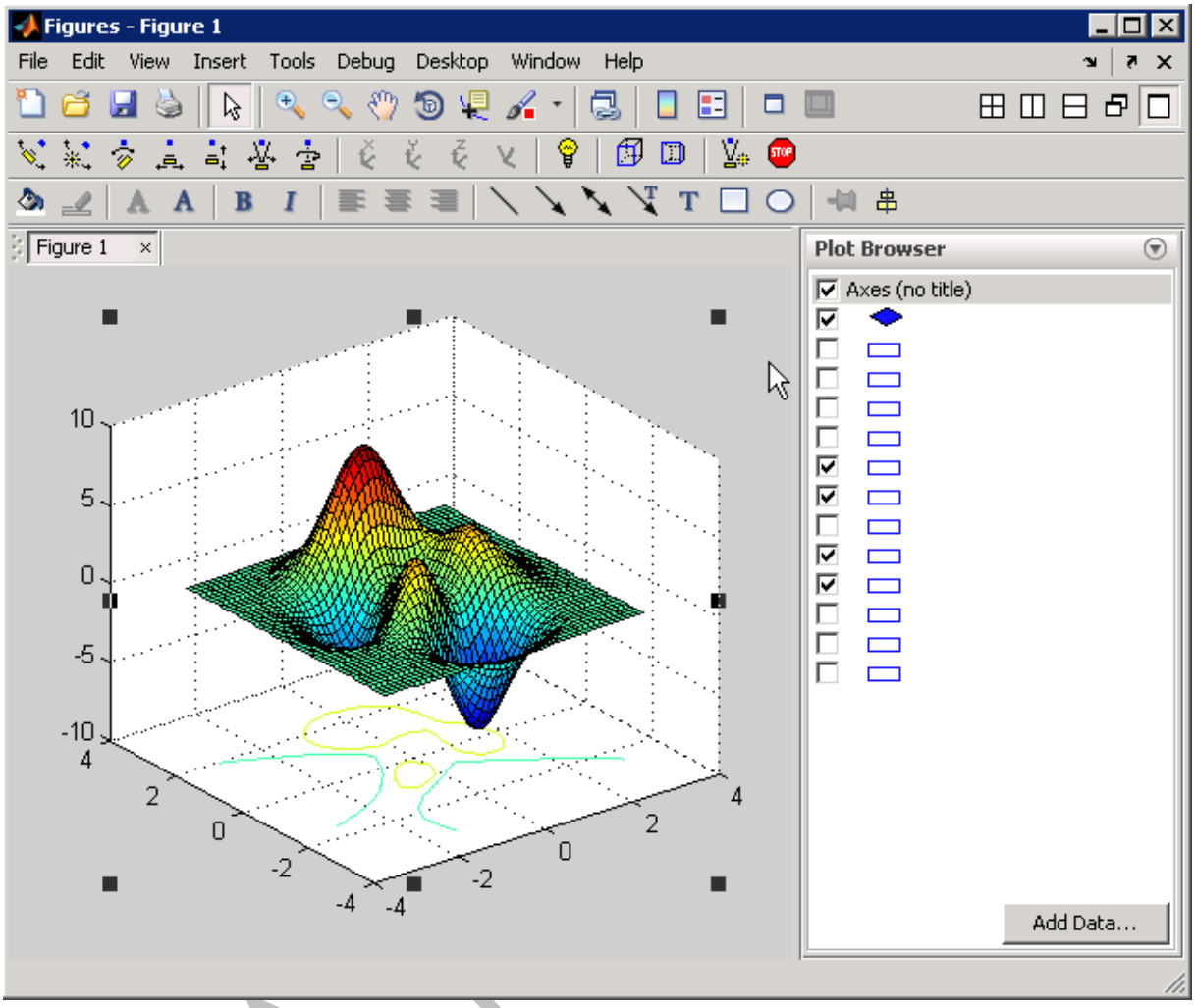
SAMPLE

This browser works in a similar way as layers for Adobe Photoshop but with a very negative point: **you can't give a name or rename the different "layer"...**



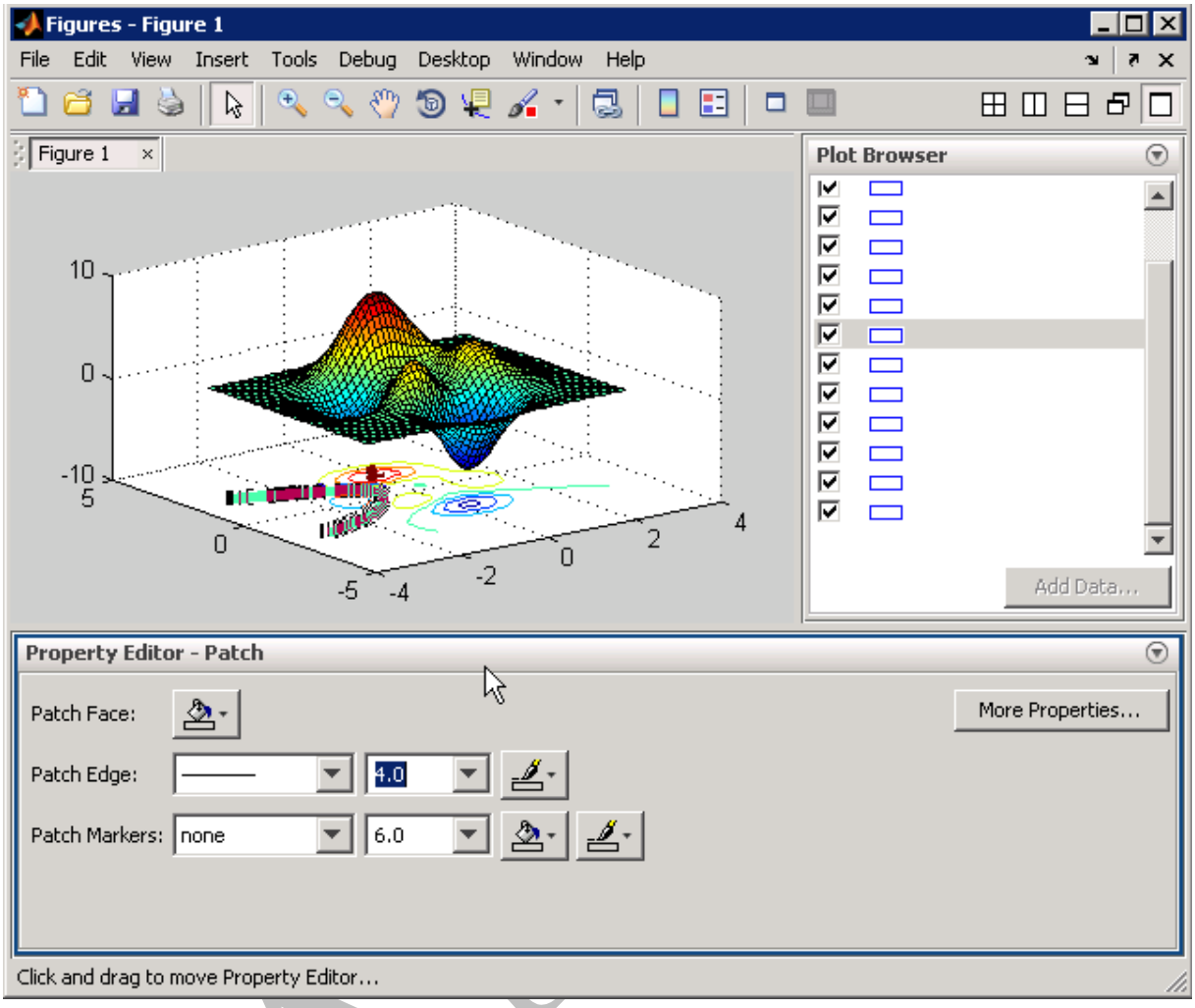
SAMPLE

Below we have for example disabled some projected isoclines:



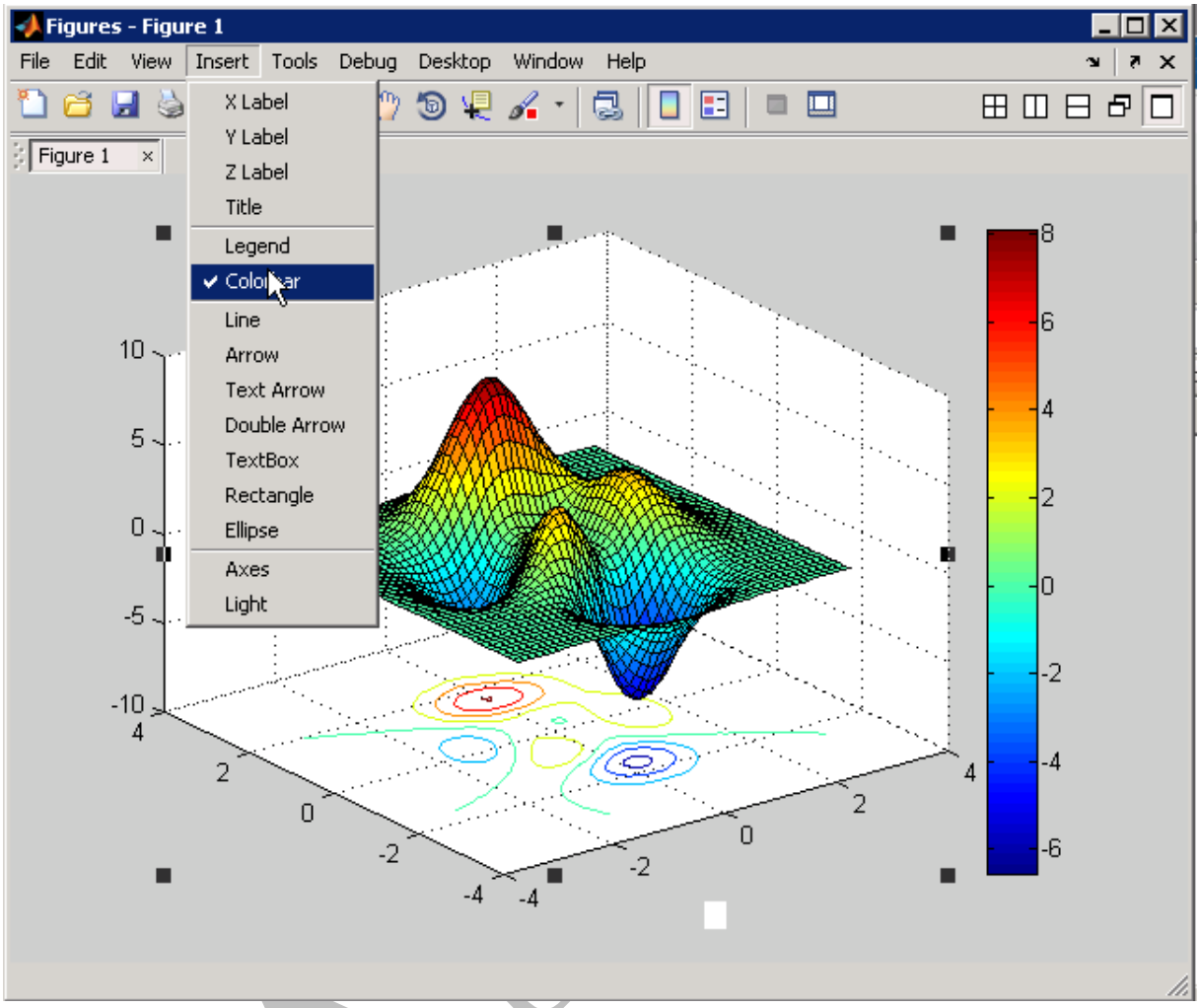
SAMPLE

By double click on one of the layers you will open the bottom **Property Editor** if it is not already activated:



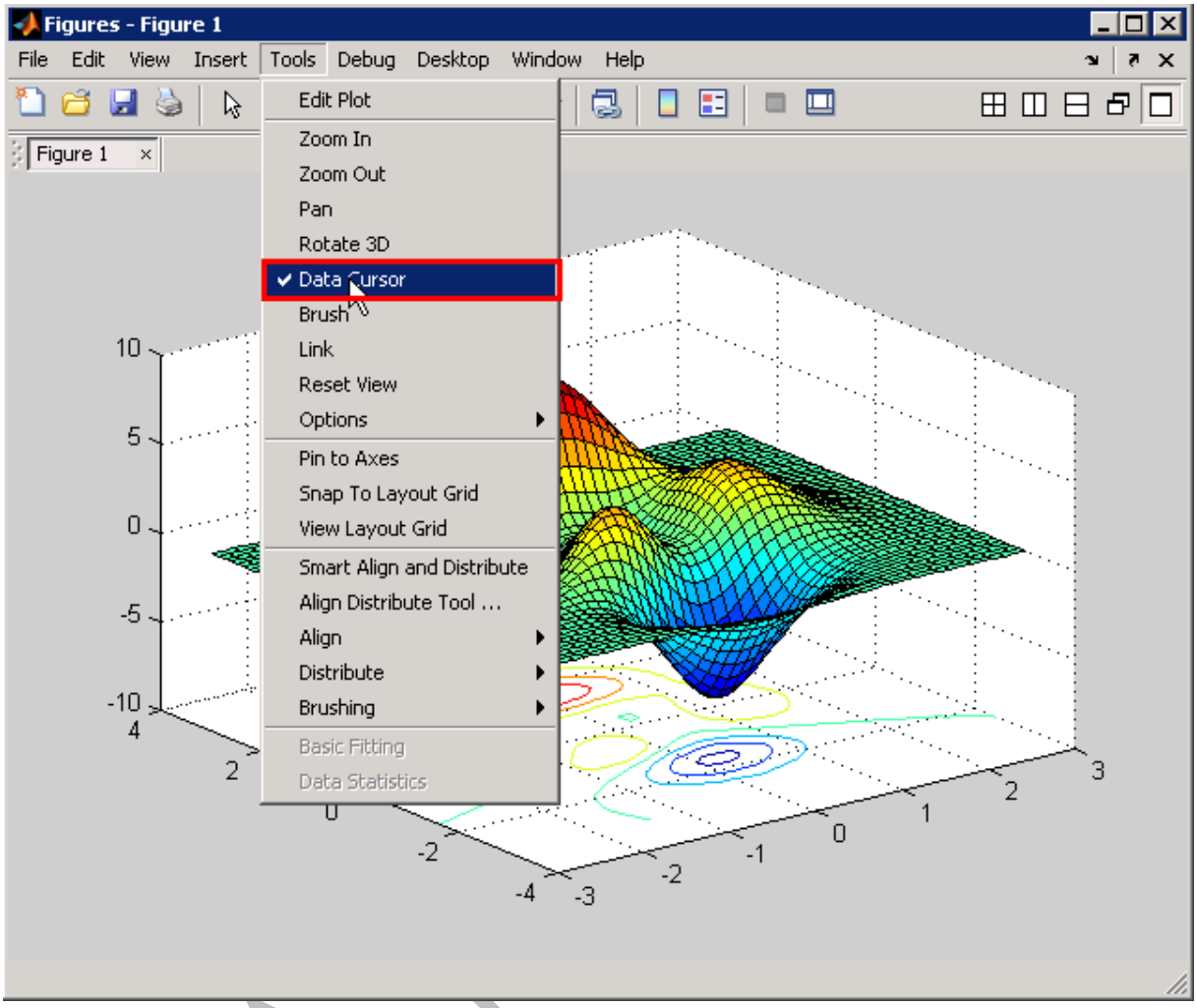
SAMPLE

In the menu **Insert** the first option that people that don't like to create chart using scripting almost always use is the **Colorbar** (the other options are too easy to guess to make examples):



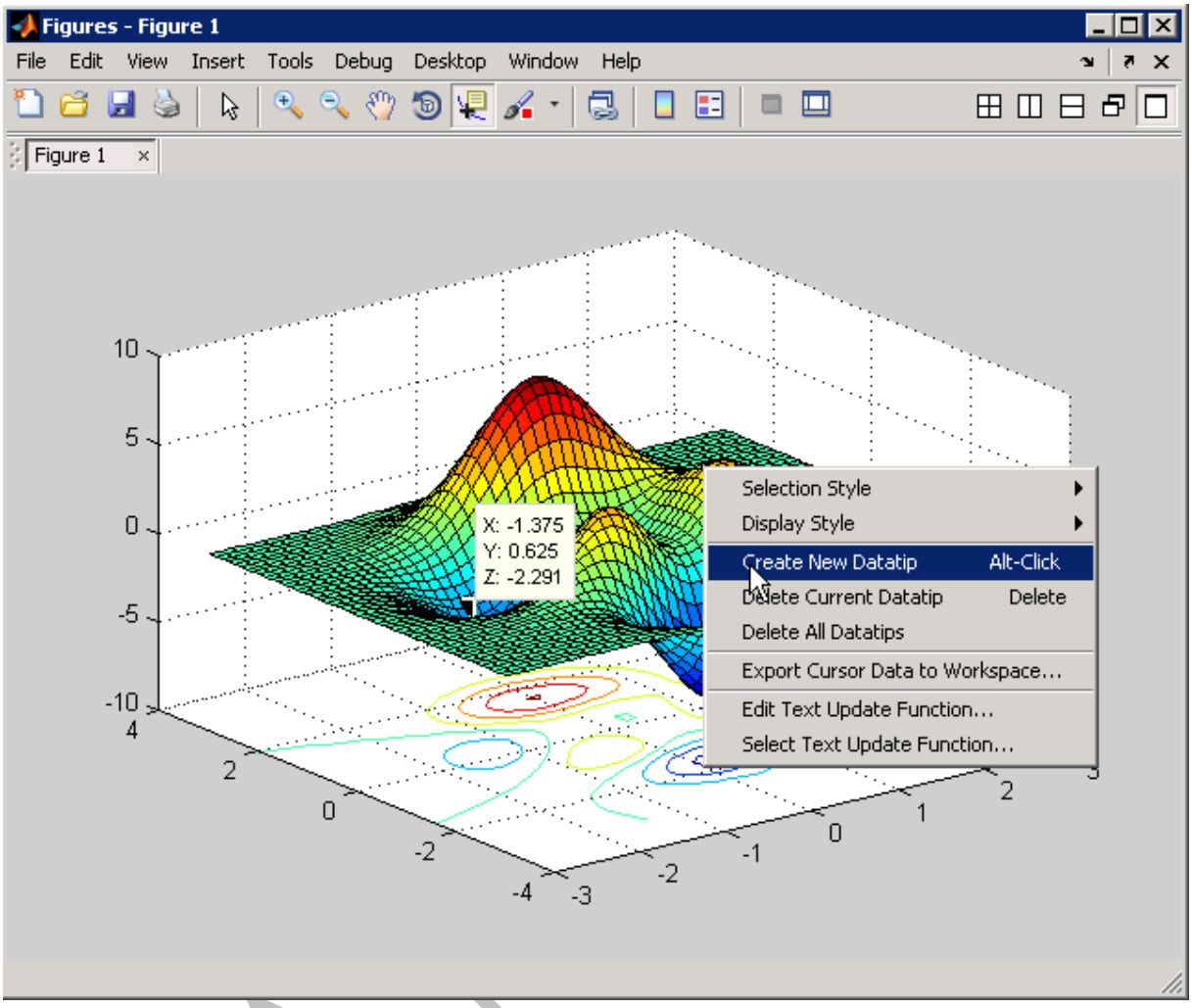
SAMPLE

Now take a look to the very useful option **Data Cursor** of the menu **Tools**:



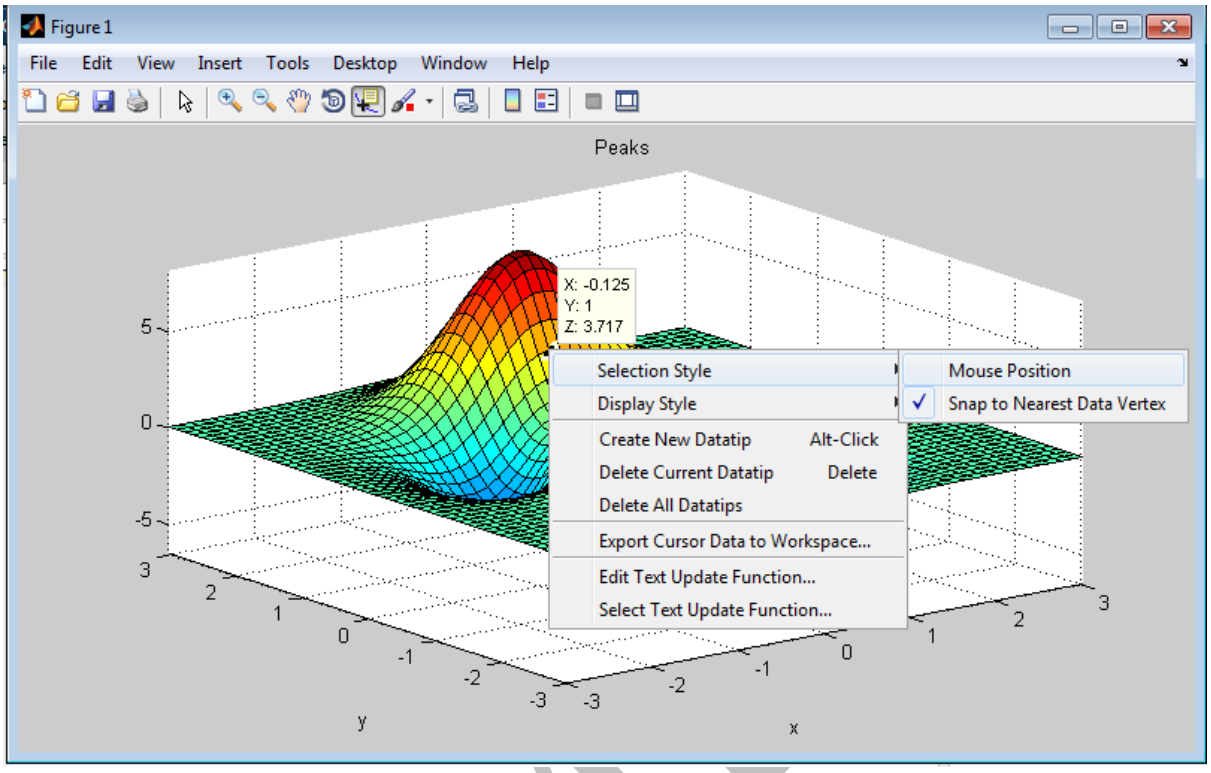
SAMPLE

This tool gives you the possibility to add data tooltips on the chart (as many as you want):

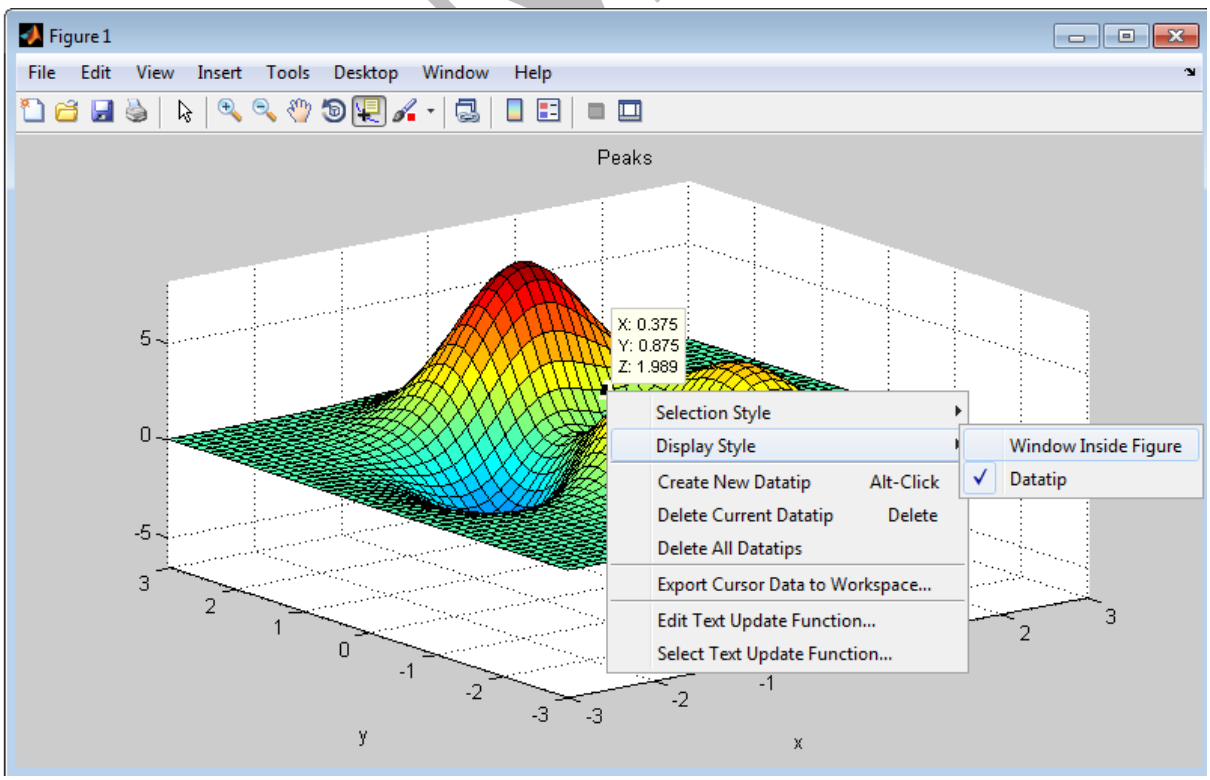


SAMPLE

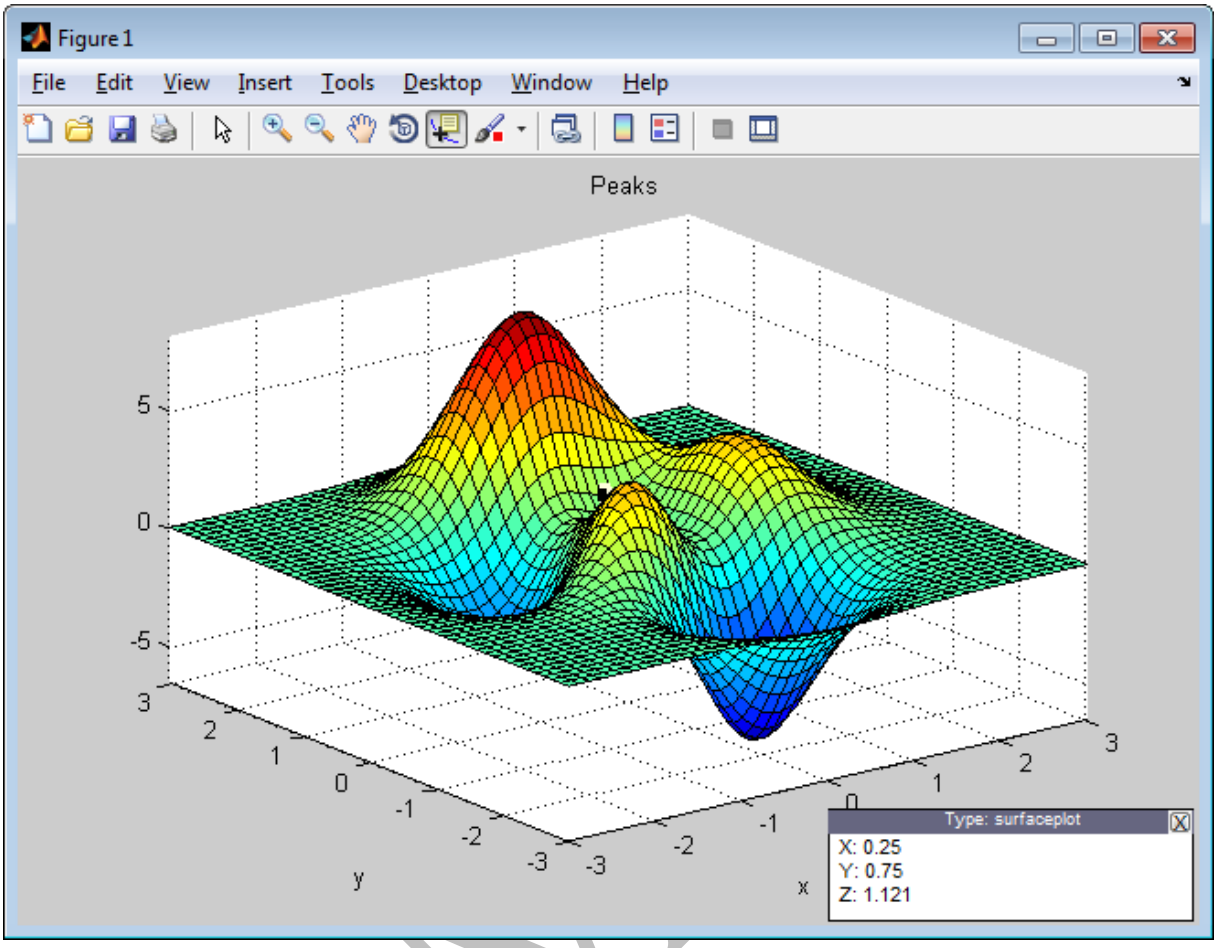
There are also a few options for these data tip. For example by default it is on Snap to **Nearest Data Vertex** but you can choose to click anywhere you want on the (surface) plot by selecting **Mouse Position**:



And there are also different ways to show the data type. You can choose for example the see the coordinates as a **Window Inside the Figure**:

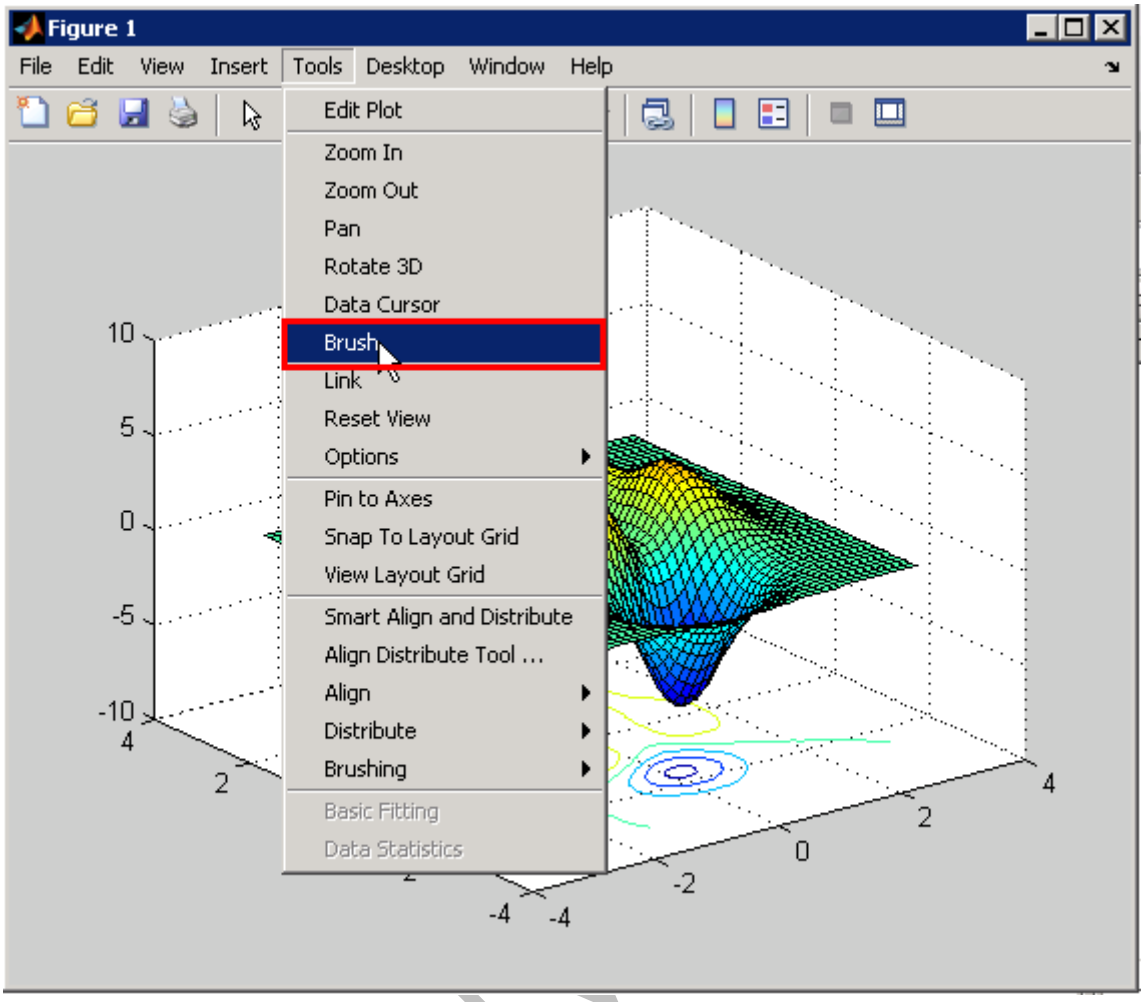


This will result to the following (see the bottom right corner):

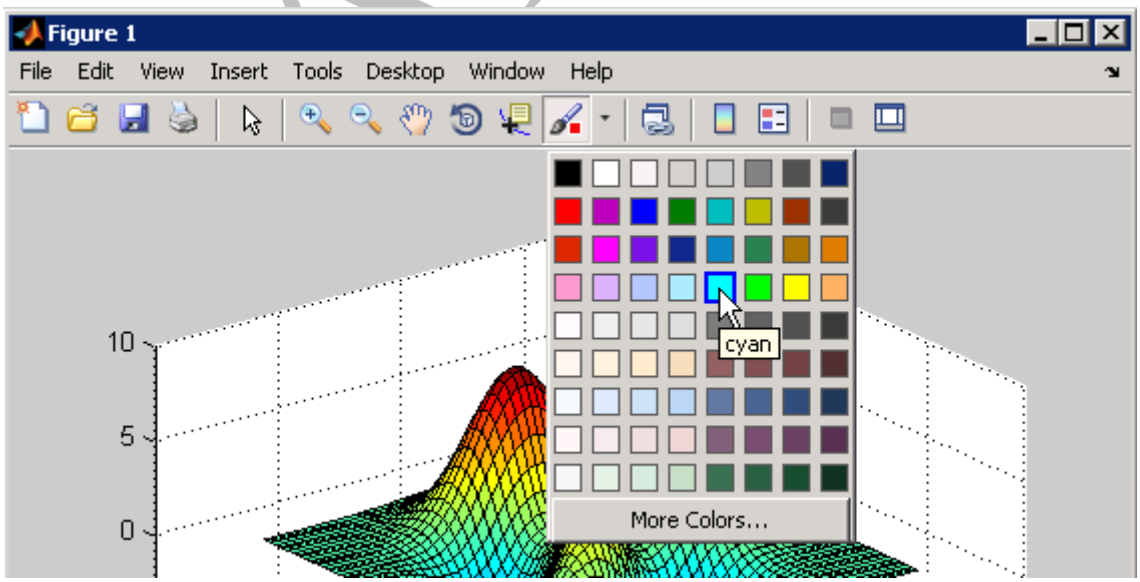


SAM

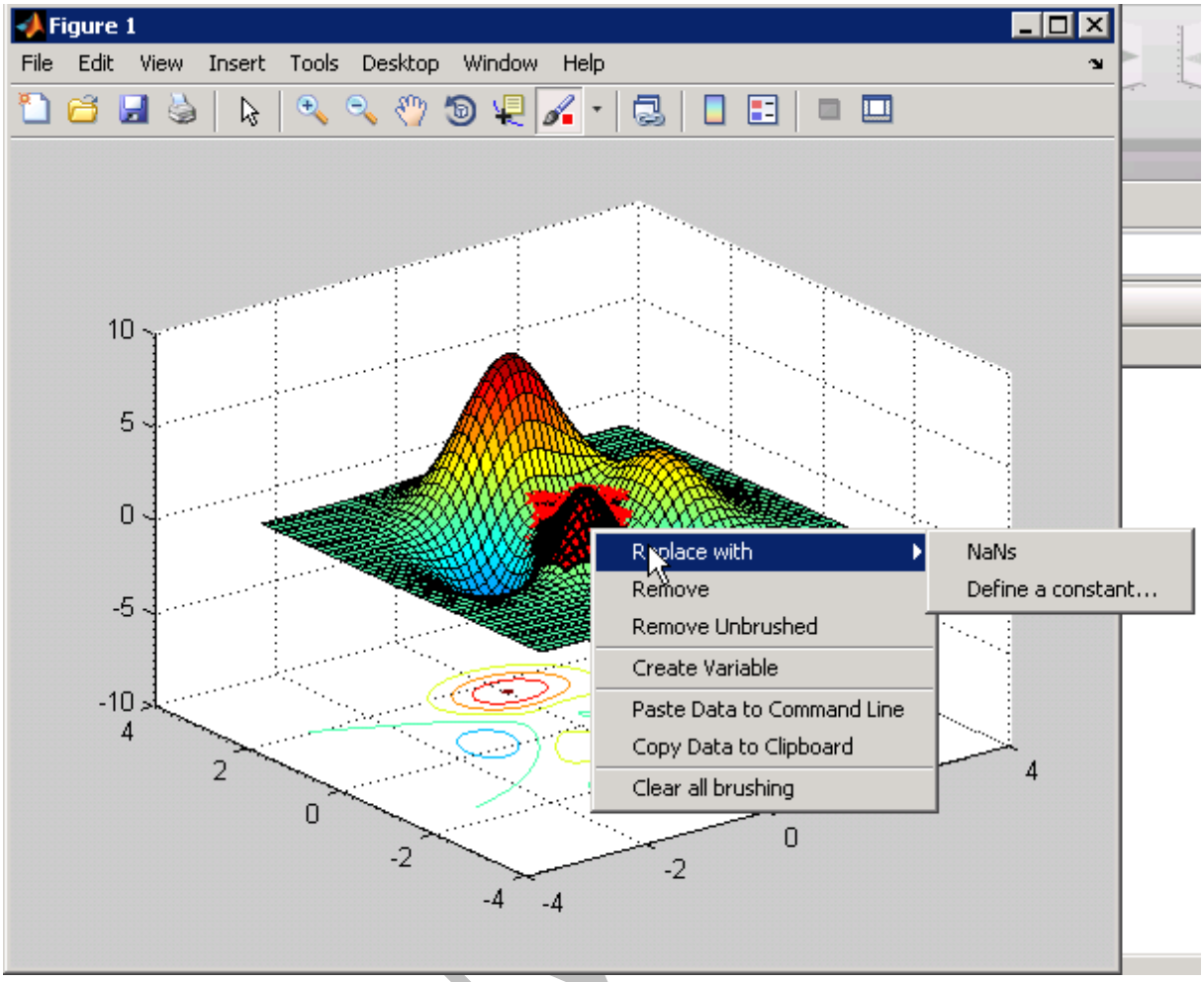
Now just take a look to the **Brush** option:



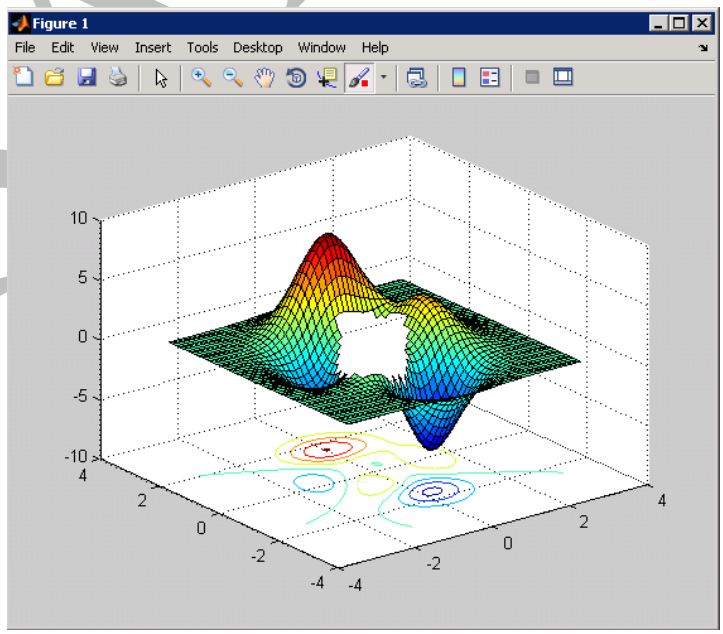
That you can also find in the main toolbar and choose a color:



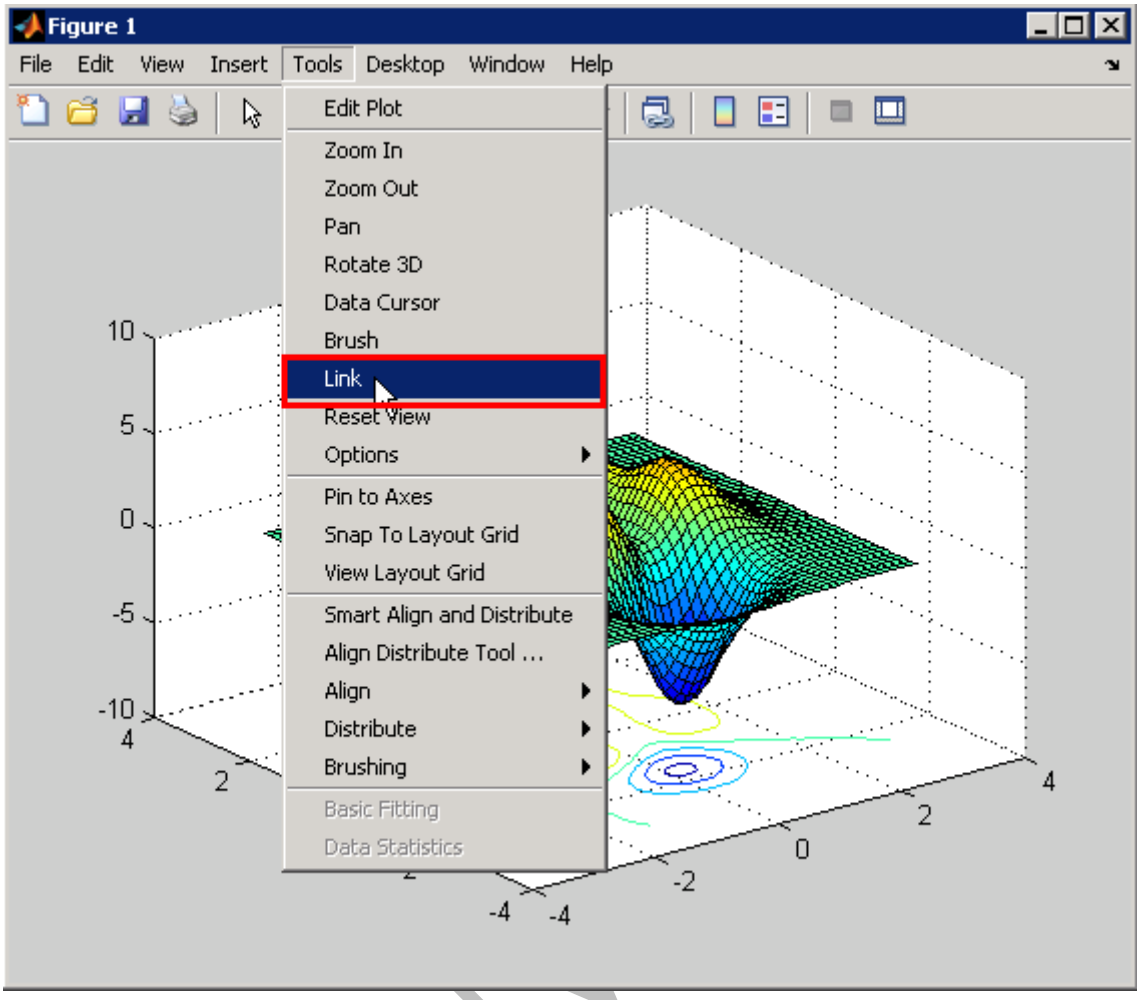
One you have finished to pain you can also make a right click and choose some option:



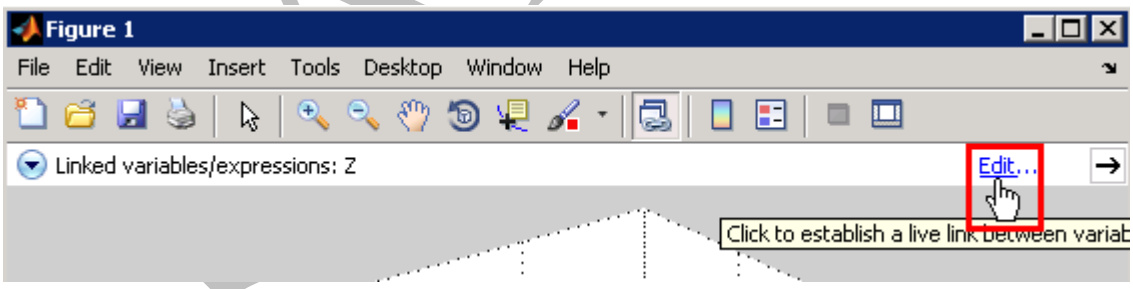
You can see below what happens when you select **Replace with/NaNs** for example:



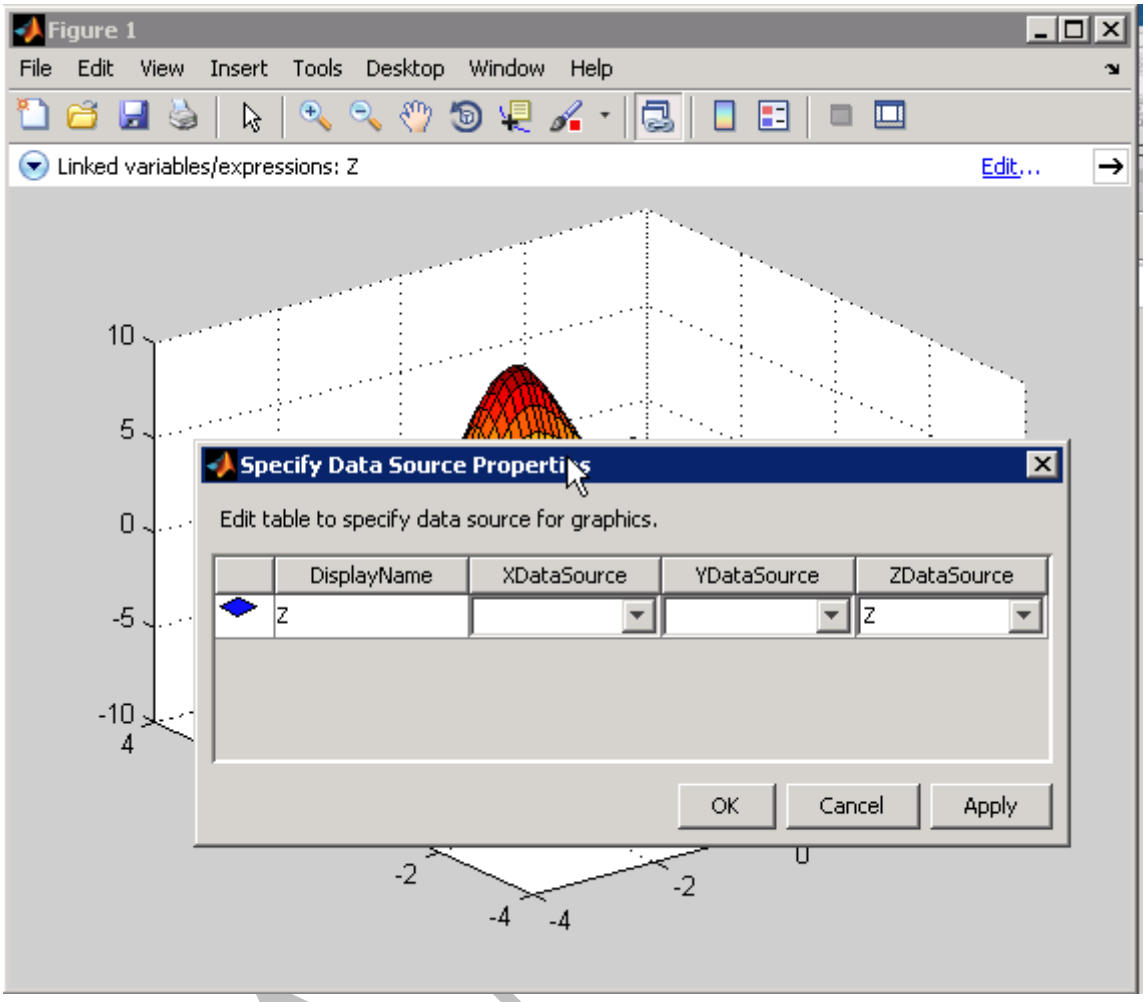
Now have a look to the option named **Link**:



You will see then a small white band appear to the top. Click on **Edit...**:

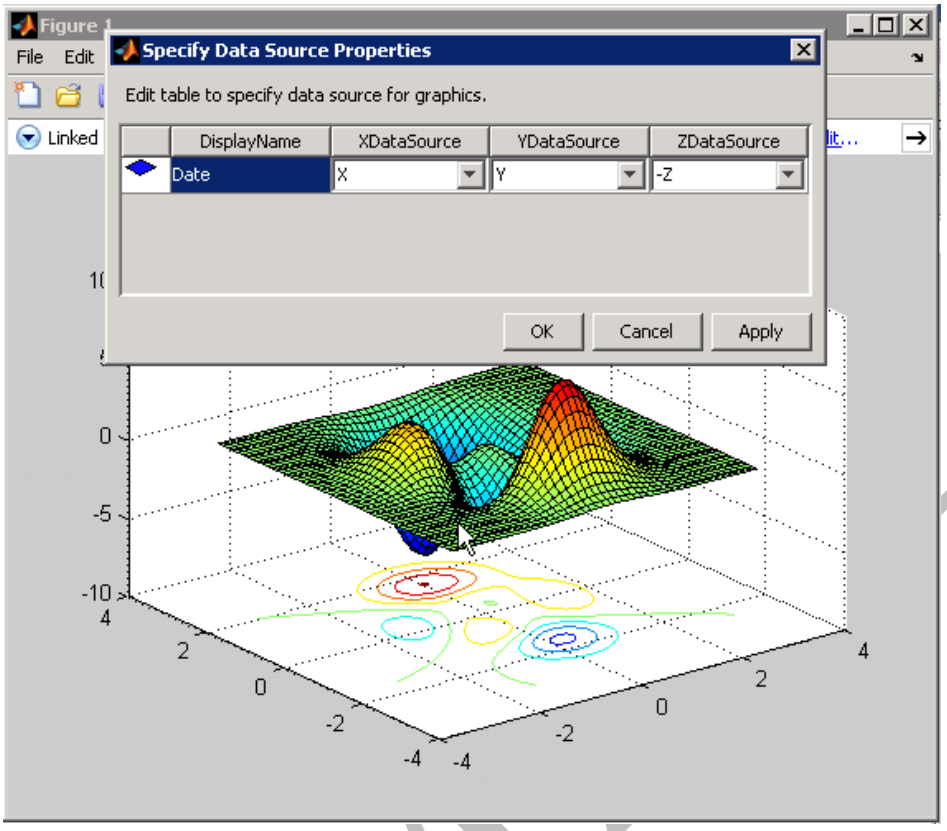


And whaa... what a surprise (it's ironic...) we just get an option to change the data source variable names to plot the chart... (same as we already saw at the beginning with this wizard):

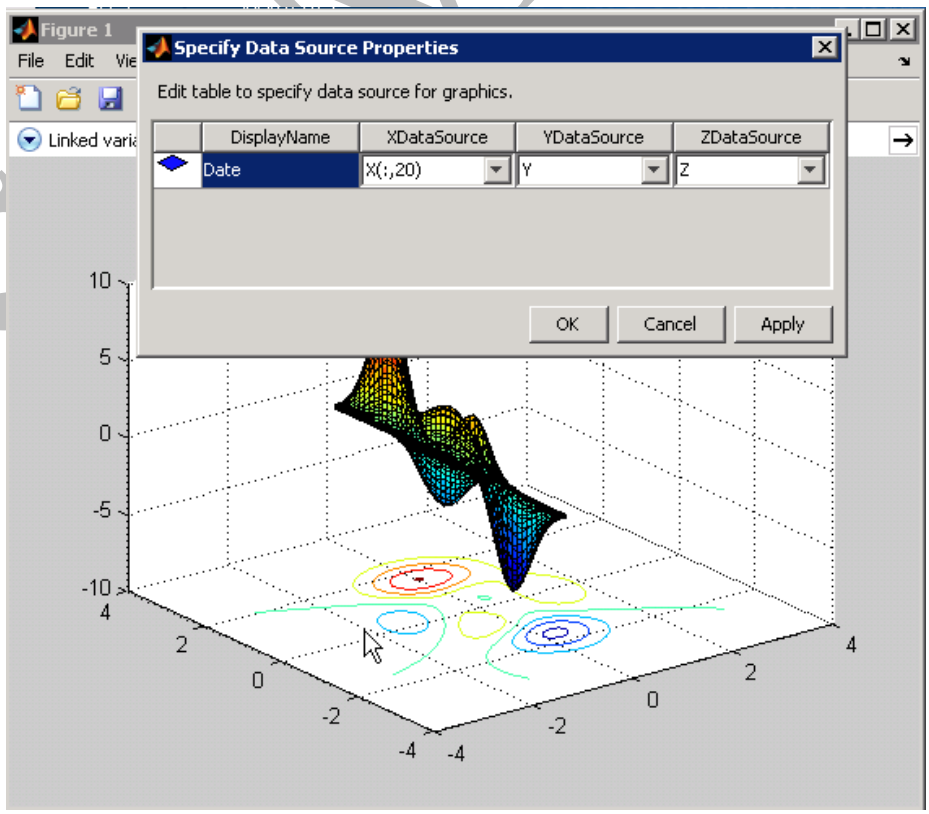


SAMPLE

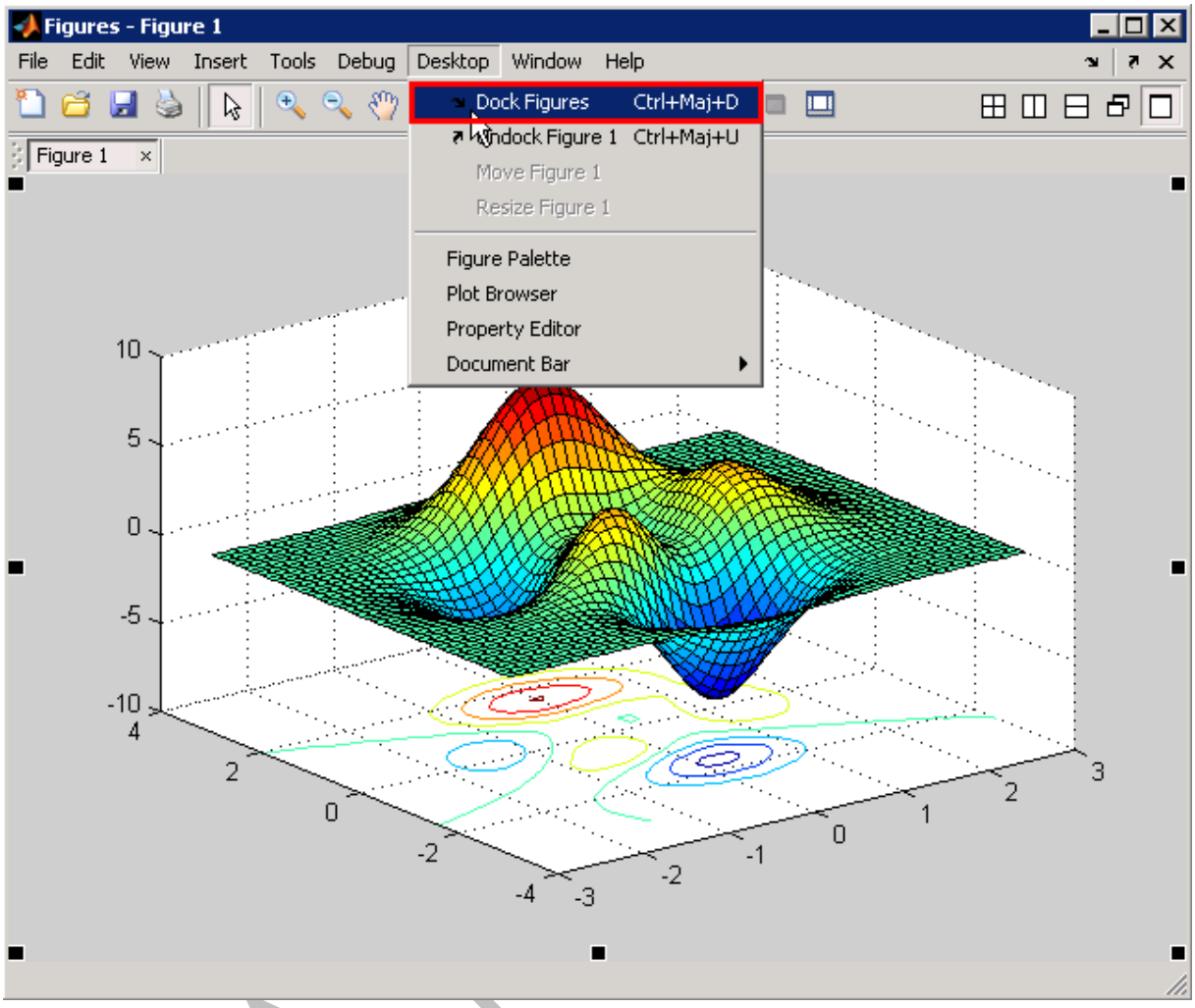
We can check for sure that it works well by defining the variables sources and playing with one of them by taking the opposite sign. Then the plotted surface will be reversed:



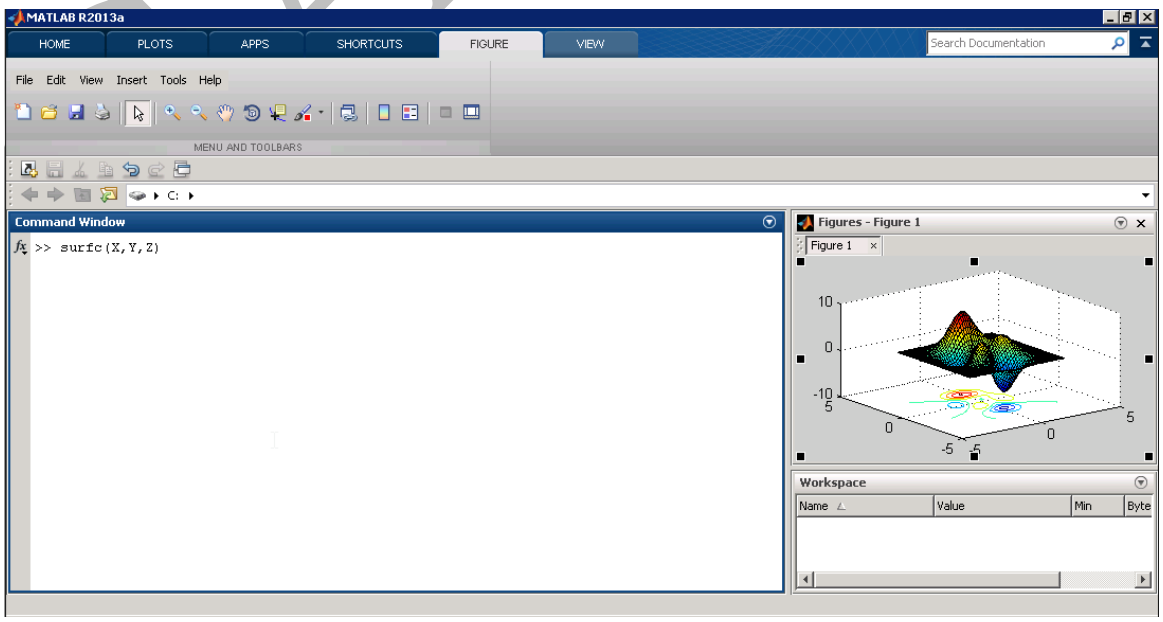
Or imagine that we are in flat world ;-) :



Now a look to the option in the menu **Desktop/Dock Figures**:



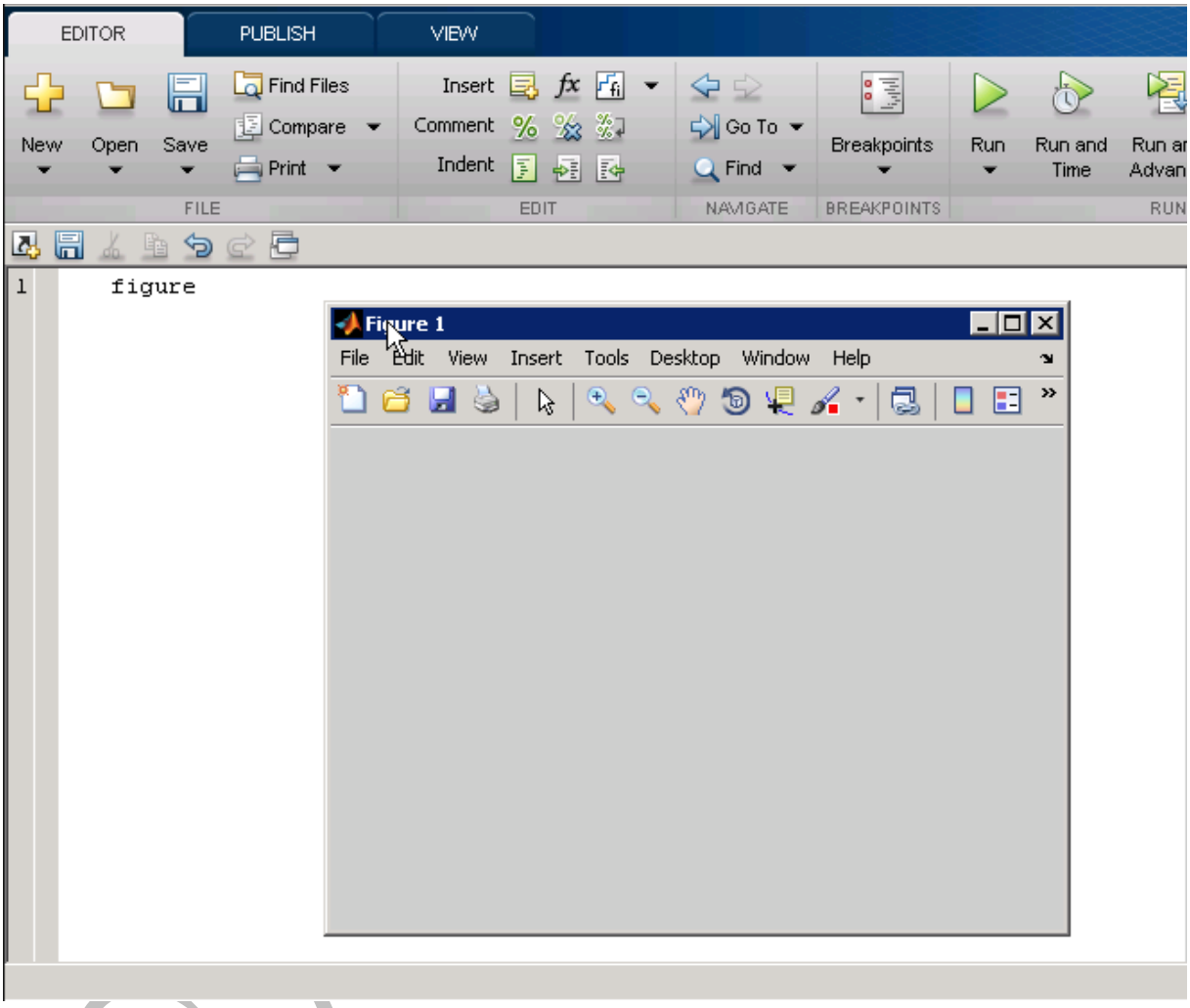
This will result in something that ca be sometimes very useful! You have the chart ant the command Windows easily always on the screen:



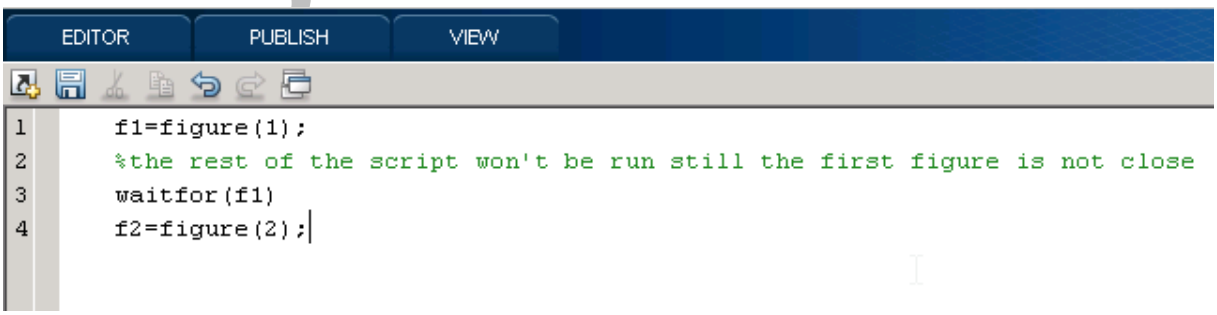
Control chart window

OK now it's time to begin to learn how to create charts and charts figure without any wizards.

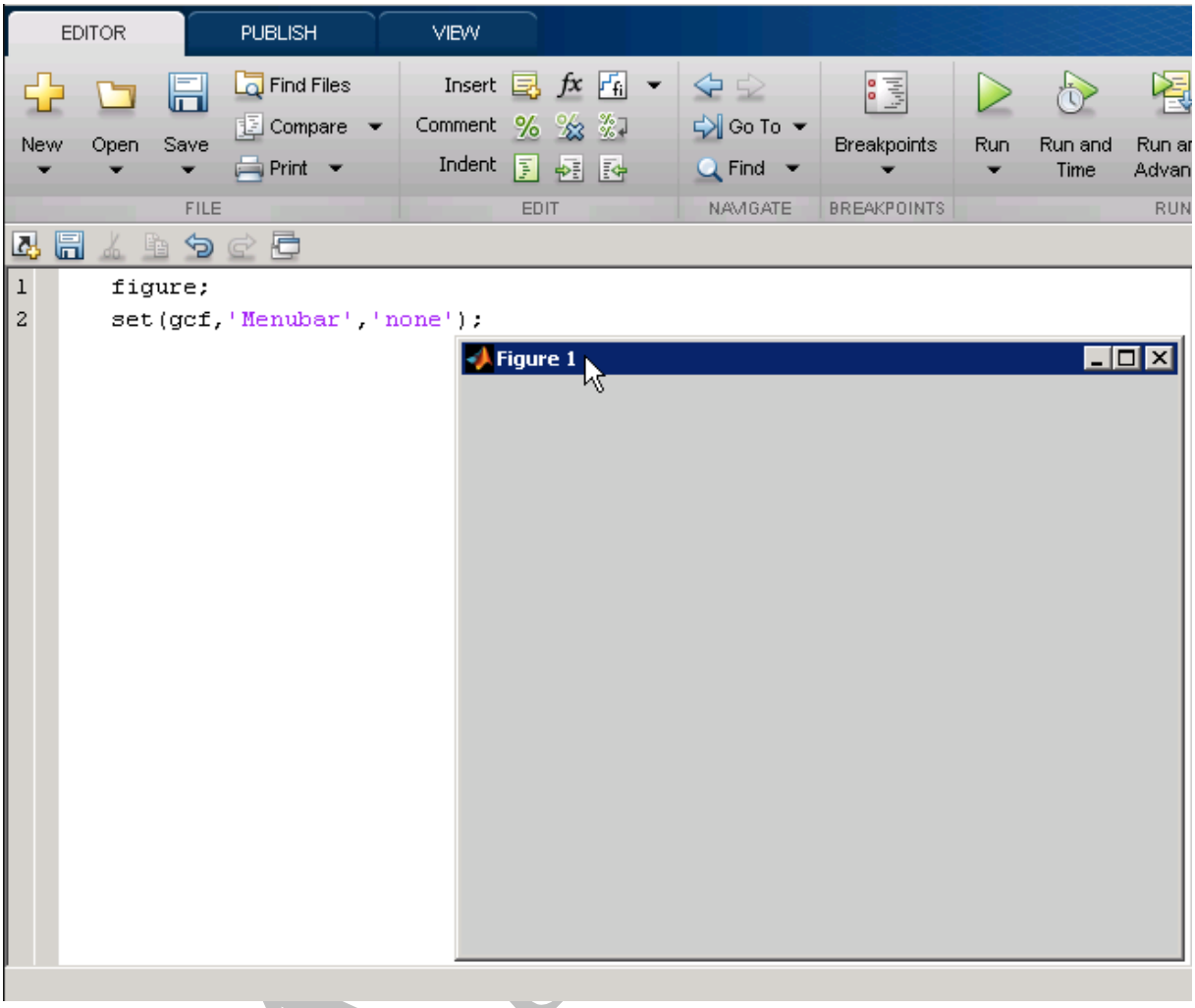
First to create an empty chart figure you can just use the command **figure** as below:



If you have to open multiple charts figure, using the function **waitfor( )** you can be sure that the rest of the script won't be run until the first figure chart is not closed:

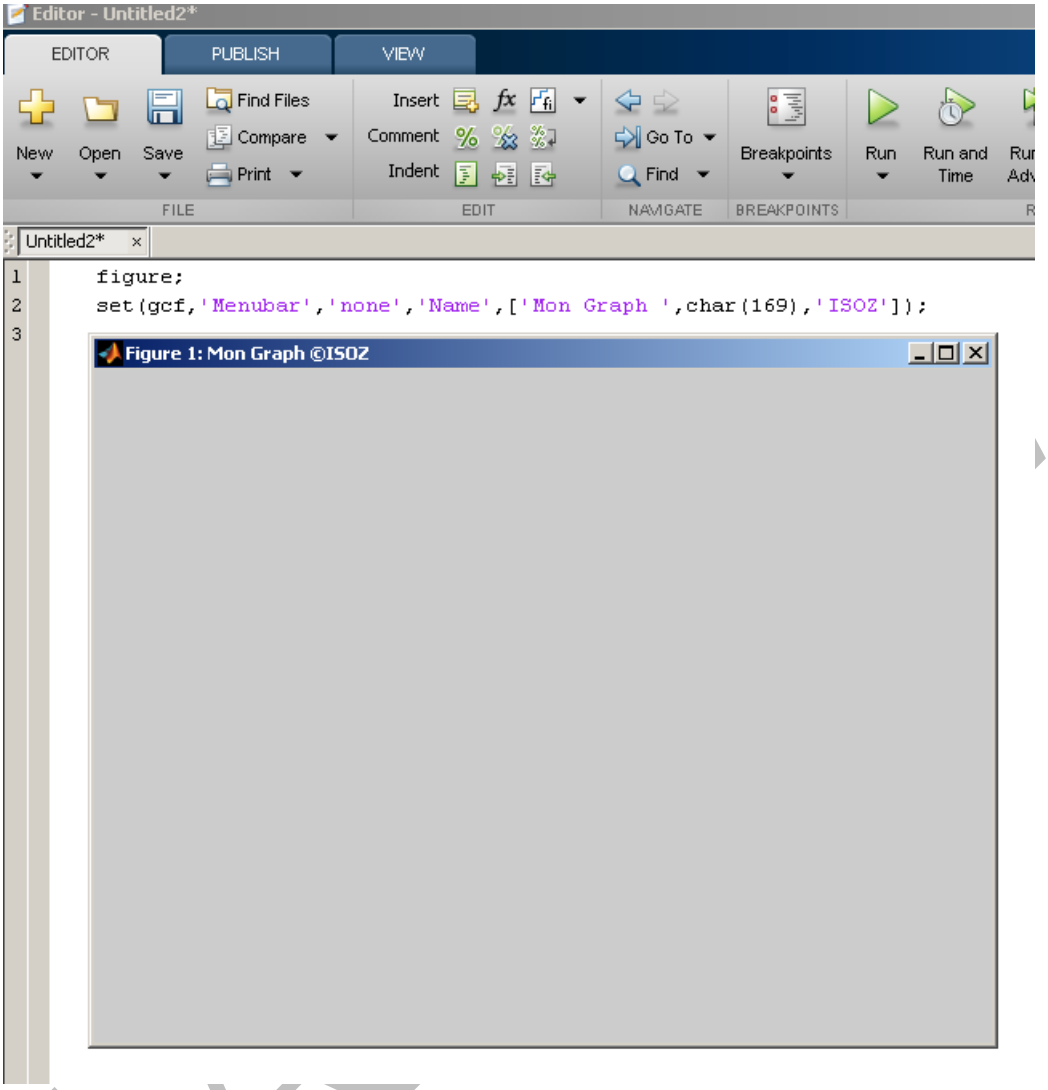


By defining some options on the active figure (**gcf**) you can remove the *Menu bar* of the figure:

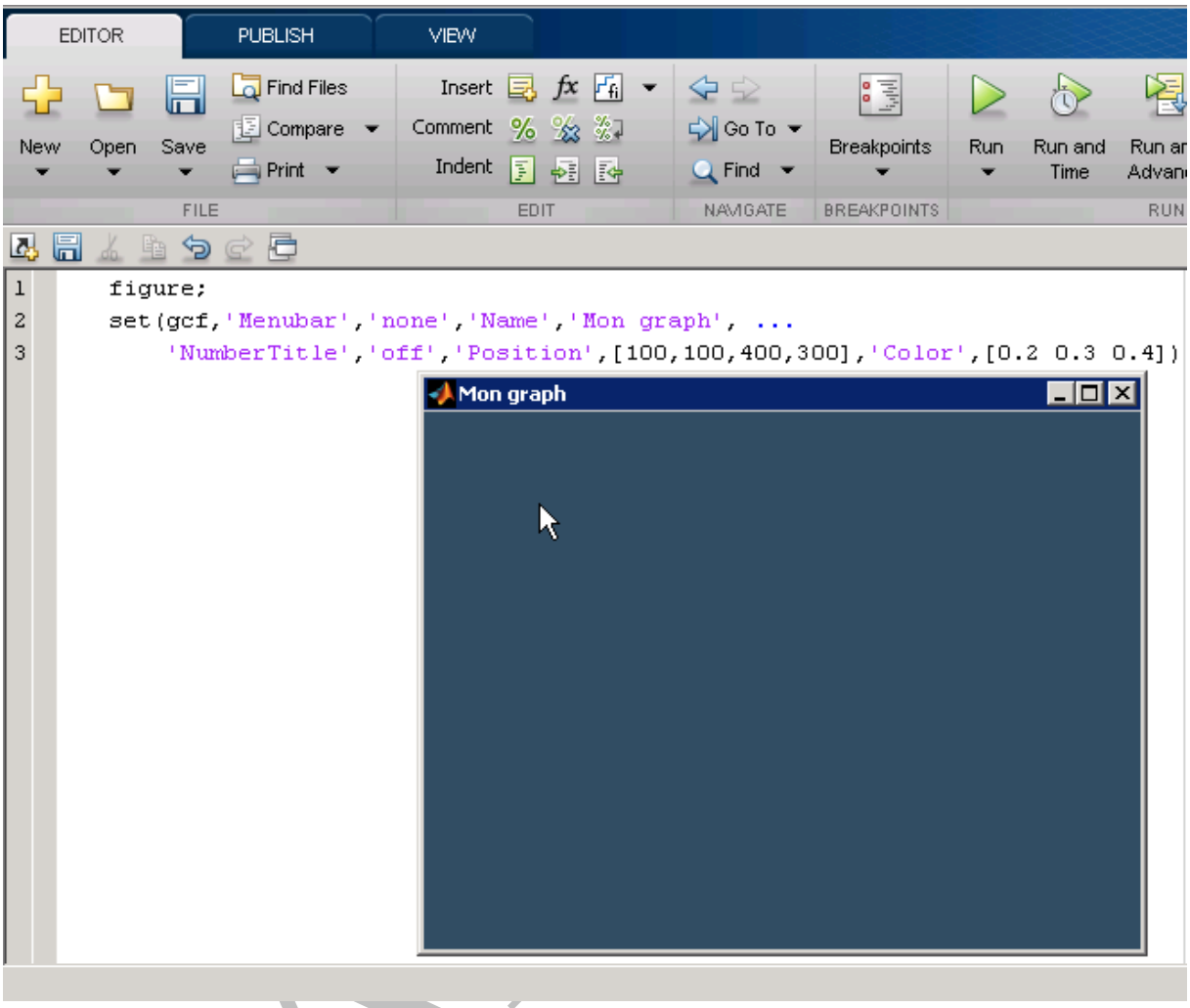


SAMPLE

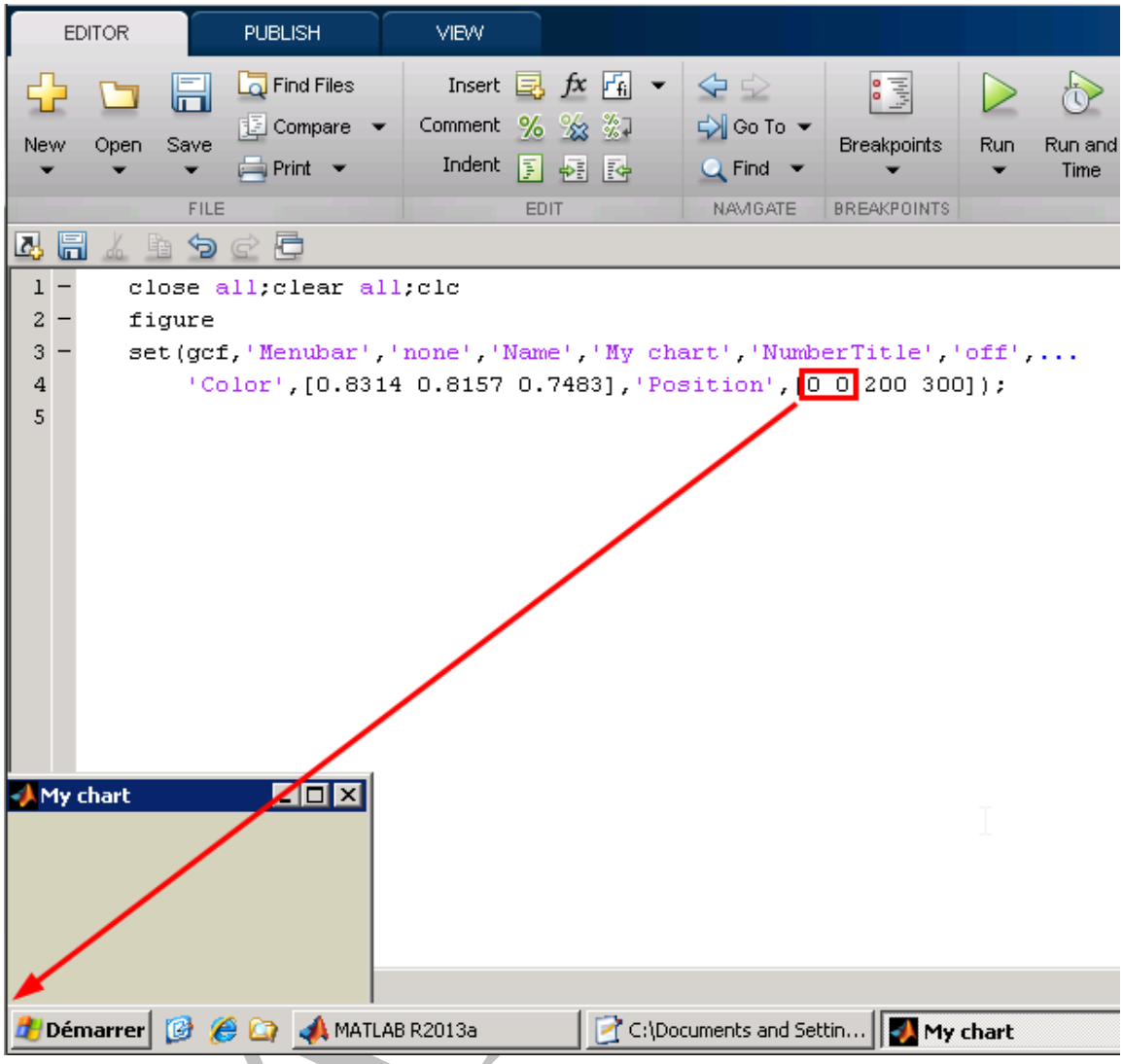
You can also with the same object properties easily define the name in the title bar of the figure:



Or also deactivate the numbering of the figure that was visible on the title bar, change it's default **Position** (the 0,0 position is in the bottom left corner!) and it's default background **Color**:



And as you can see it here as a proof, the 0,0 position is in the bottom left corner:

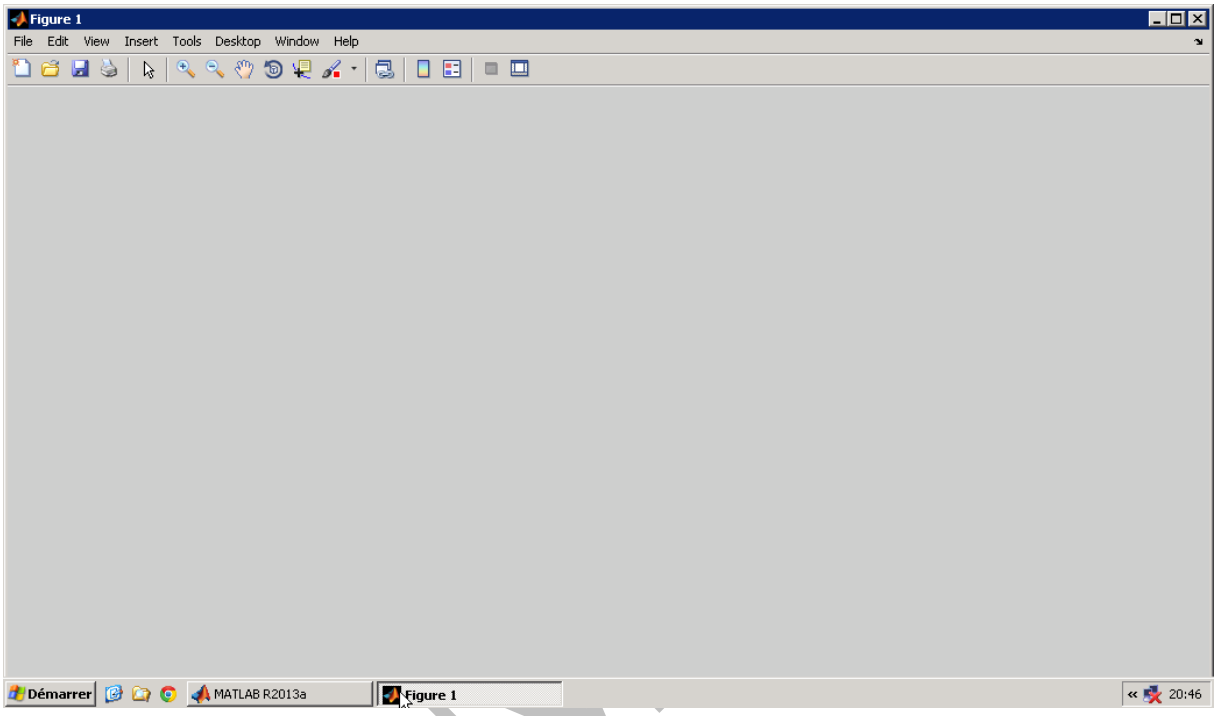


SA

We can also make a full screen chart window using the following syntax:

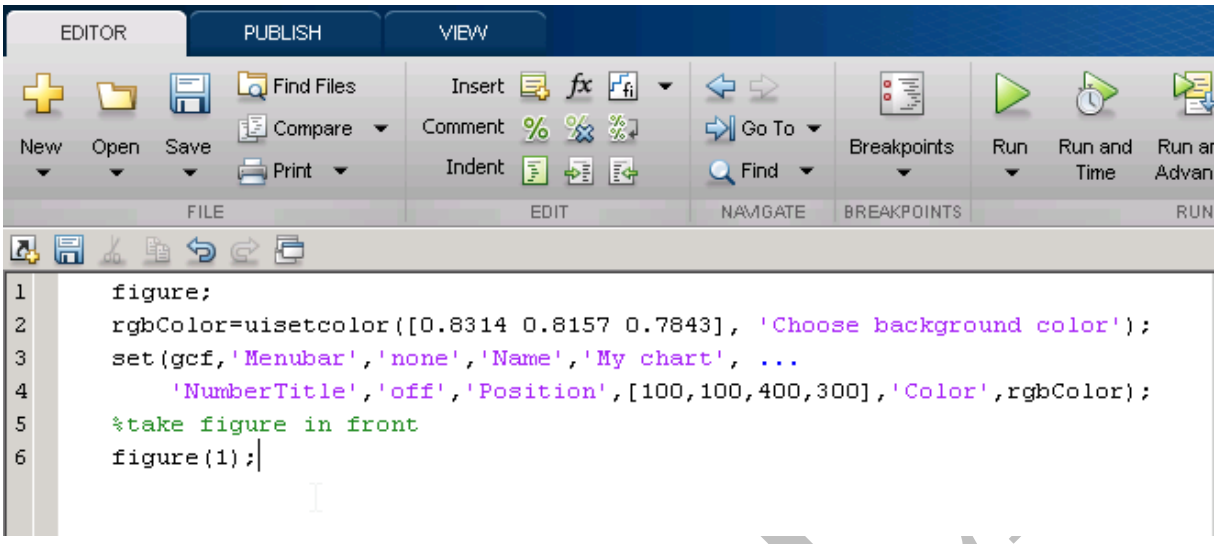
```
figure('units','normalized','outerposition',[0 0 1 1])
```

Then we get:

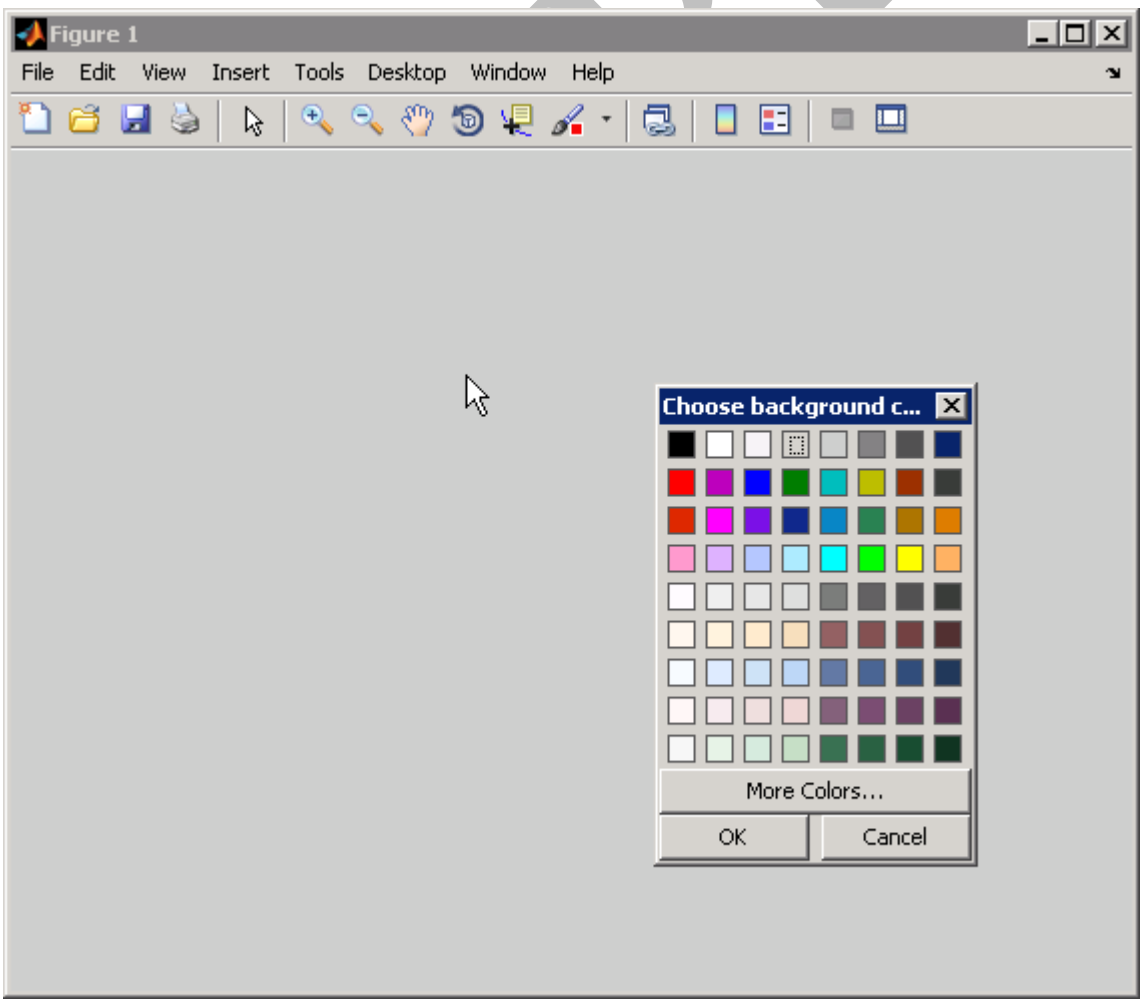


SAM

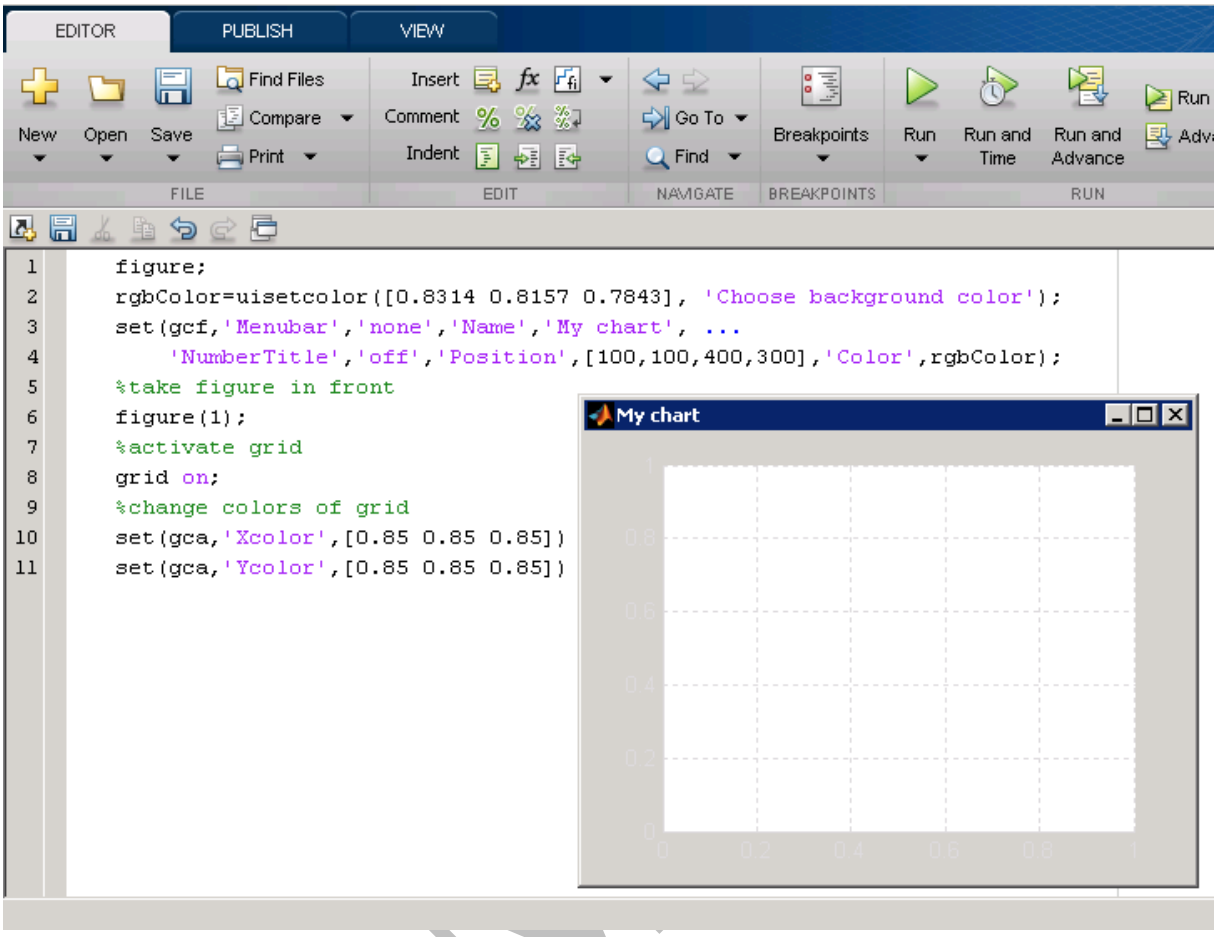
Using what we learned about User Interactions (see page 200), we can also ask the user to choose any color with the **uisetcolor** function (in this case with a recommended color activated by default):



When you run the script the result will looks like this:



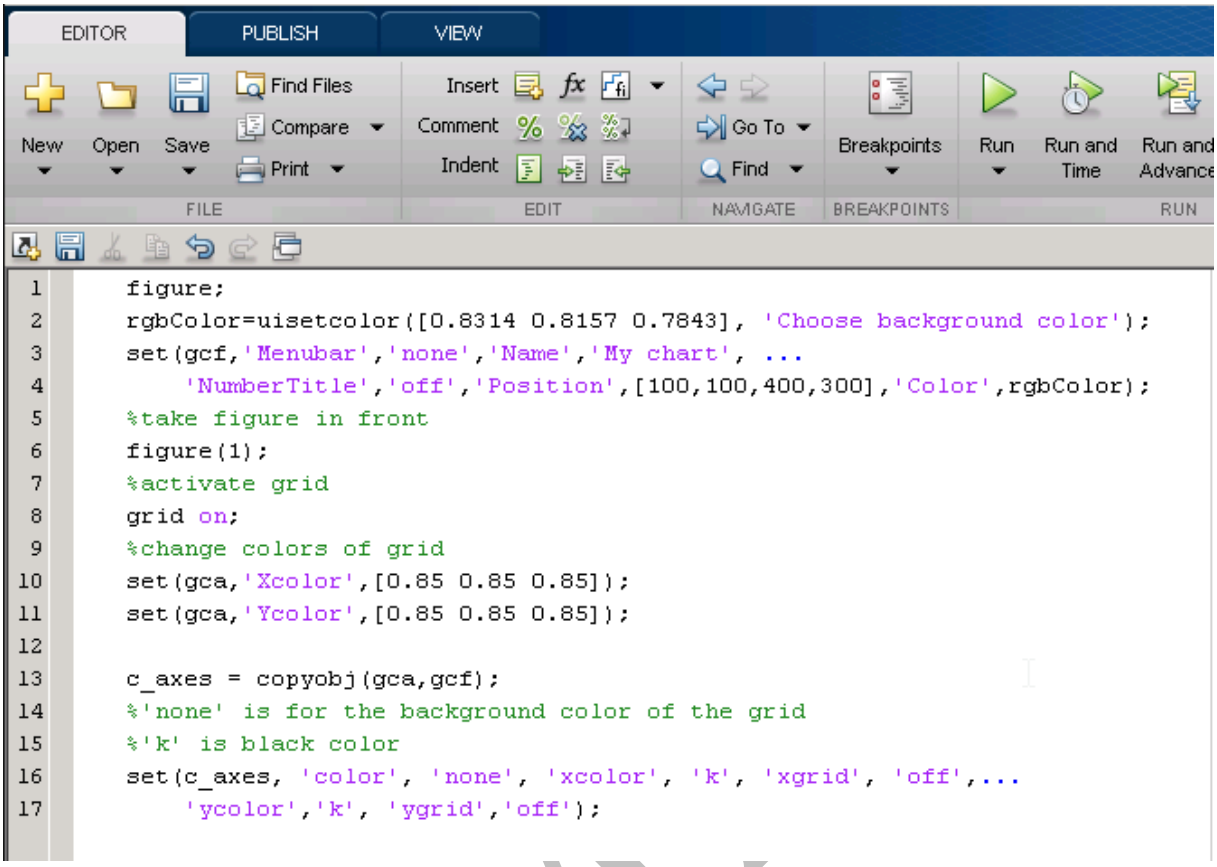
Now let us put a grid, using the command **grid on** and at the same time changing the grid color thanks to the object **gca**:



The problem that when we customize the grid, we also change the axes ticks and labels in the same color... thus see next page what we have to add:

SAMPLE

Here is what we have to add to recover back the axes in a more common color...:




```
figure;
rgbColor=uisetcolor([0.8314 0.8157 0.7843], 'Choose background color');
set(gcf,'Menubar','none','Name','My chart', ...
    'NumberTitle','off','Position',[100,100,400,300],'Color',rgbColor);
%take figure in front
figure(1);
%activate grid
grid on;
%change colors of grid
set(gca,'Xcolor',[0.85 0.85 0.85]);
set(gca,'Ycolor',[0.85 0.85 0.85]);

c_axes = copyobj(gca,gcf);
%'none' is for the background color of the grid
%'k' is black color
set(c_axes, 'color', 'none', 'xcolor', 'k', 'xgrid', 'off',...
    'ycolor','k', 'ygrid','off');
```

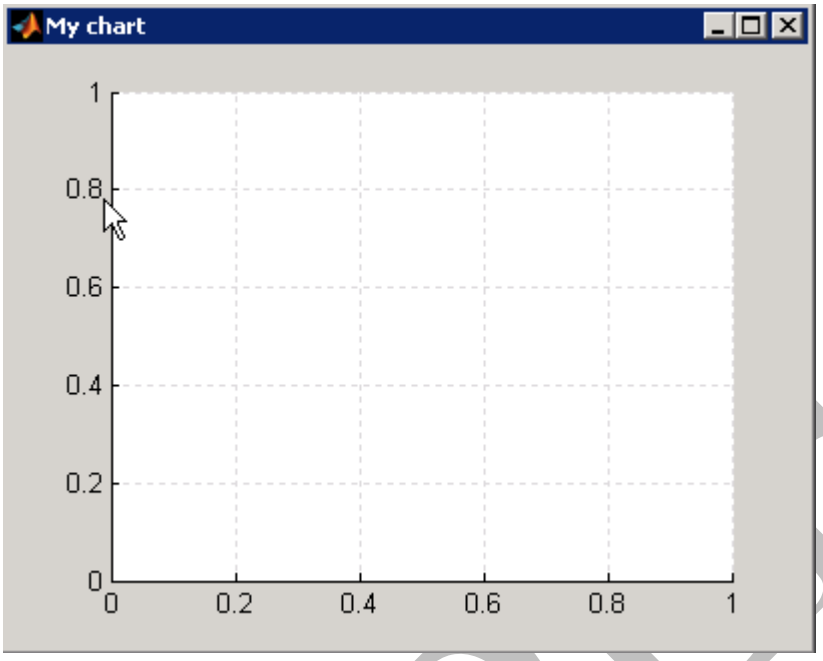
Not very intuitive isn't it? The result is visible on the next page.

For the colors here is a small table as refresh:

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

Table 1 Standardized color codes

Here is the result:

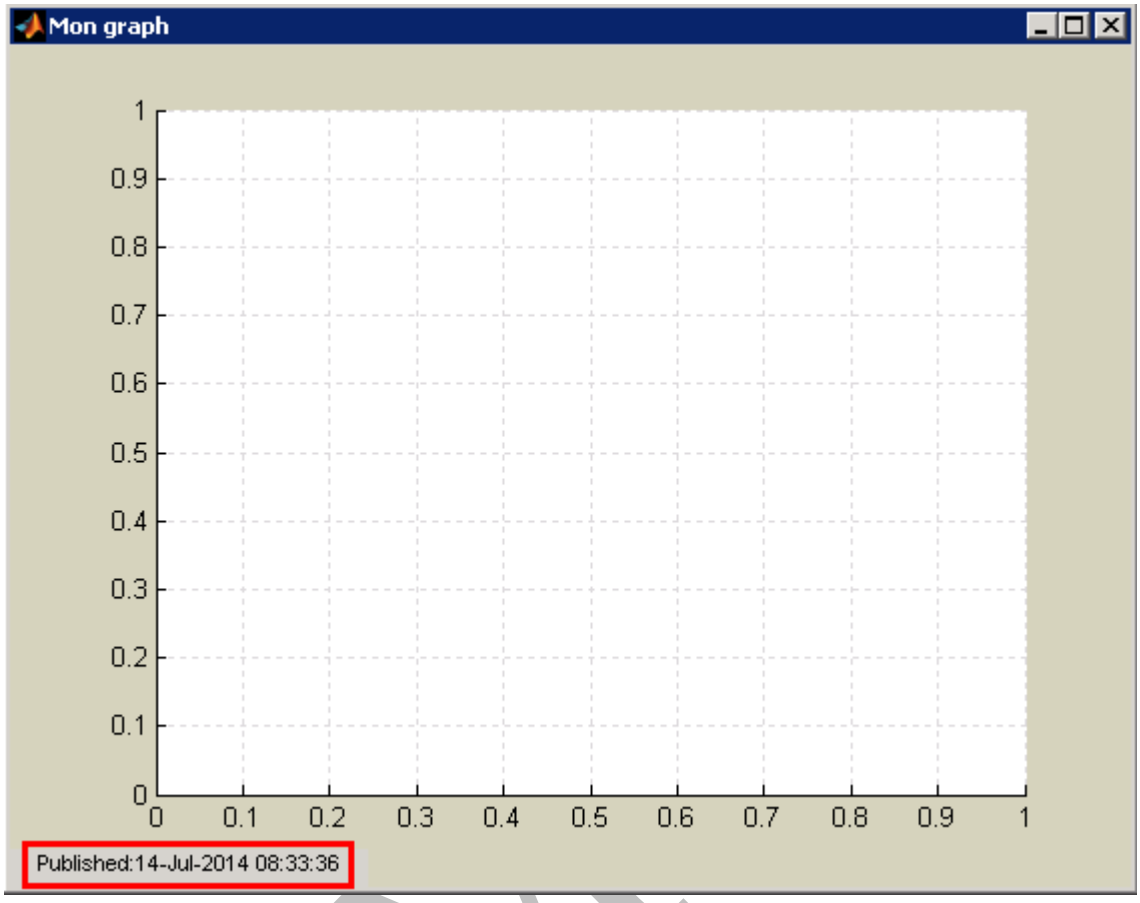


Now a very common request by engineers is to put like in Minitab the actual date and time (or any other date) in the corner of the chart figure. We can do this using a text type **uicontrol**:

```
EDITOR  PUBLISH  VIEW

1 -  close all;clear all;clc
2 -  figure
3 -  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4        'Color',[0.8314 0.8157 0.7483]);
5
6 -  grid on;
7
8 -  set(gca,'Xcolor',[0.85 0.85 0.85]);
9 -  set(gca,'Ycolor',[0.85 0.85 0.85]);
10 -  setAxes=copyobj(gca,gcf);
11 -  set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
12 -  MyBox=uicontrol('style','text');
13 -  set(MyBox,'String',strcat('Published: ',datestr(clock)),'Position',[0, 0, 180, 20])
14
```

The result will look like the following:



SAM

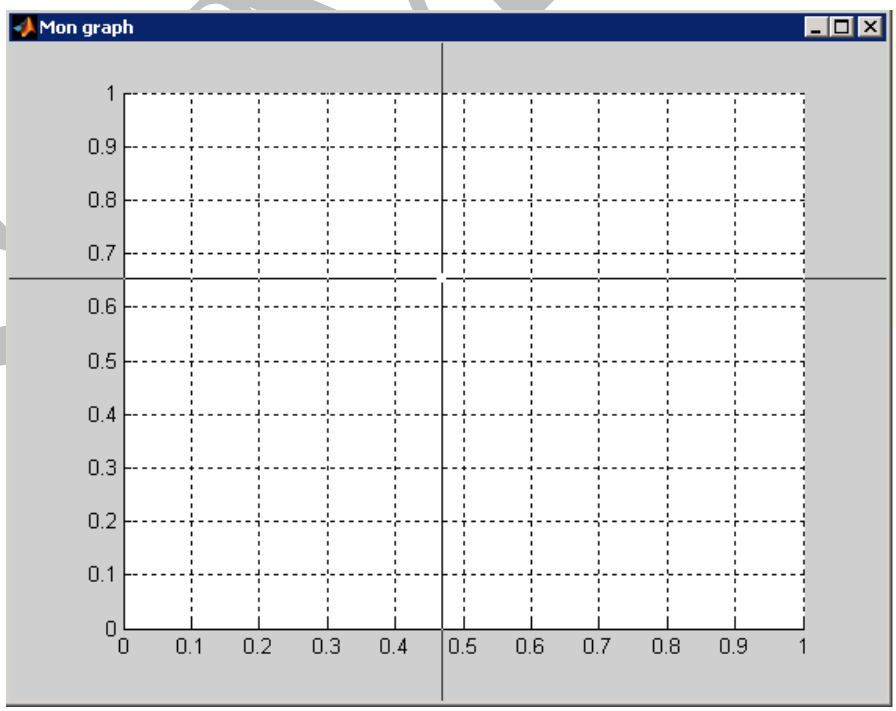
Pick-up points on chart

We have removed all the menus and toolbar of our chart figure then a very common question is how we can easily and efficiently for post analysis make that the user can click on the chart an retrieve coordinates in the workspace.

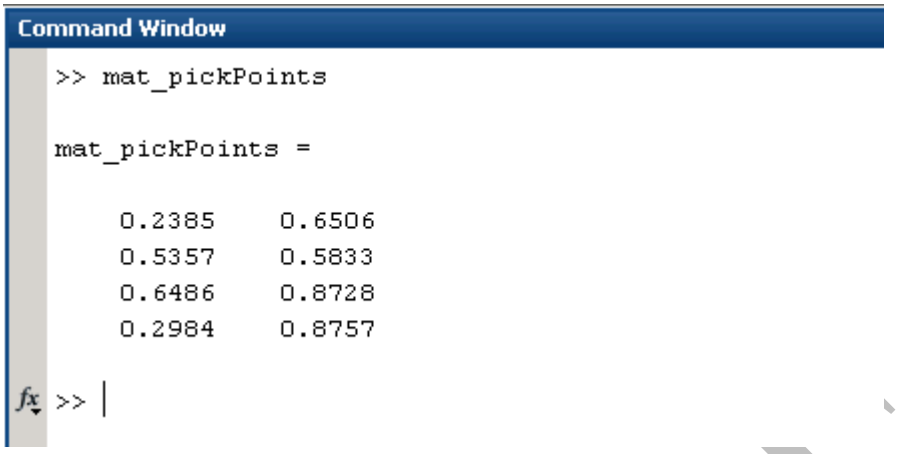
For this purpose you can use the **ginput( )** function by specifying the number of points to retrieve in the workspace:

```
1   figure
2   set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
3       'Color',[0.8 0.8 0.8])
4
5   grid on;
6   %get 4 points
7   mat_pickPoints=ginput(4);
8
9   set(gca,'Xcolor',[0.85 0.85 0.85])
10  set(gca,'Ycolor',[0.85 0.85 0.85])
11  c_axes=copyobj(gca,gcf);
12  set(c_axes,'color','none','xcolor','k','xgrid','off','ycolor','k','ygrid','off');
```

This will result in a snipper target tool to pick up the coordinates as you can see below on this empty chart:



And you will get in the workspace a new matrix variable with all coordinates that you can use for multiple purposes:



SAMPLE

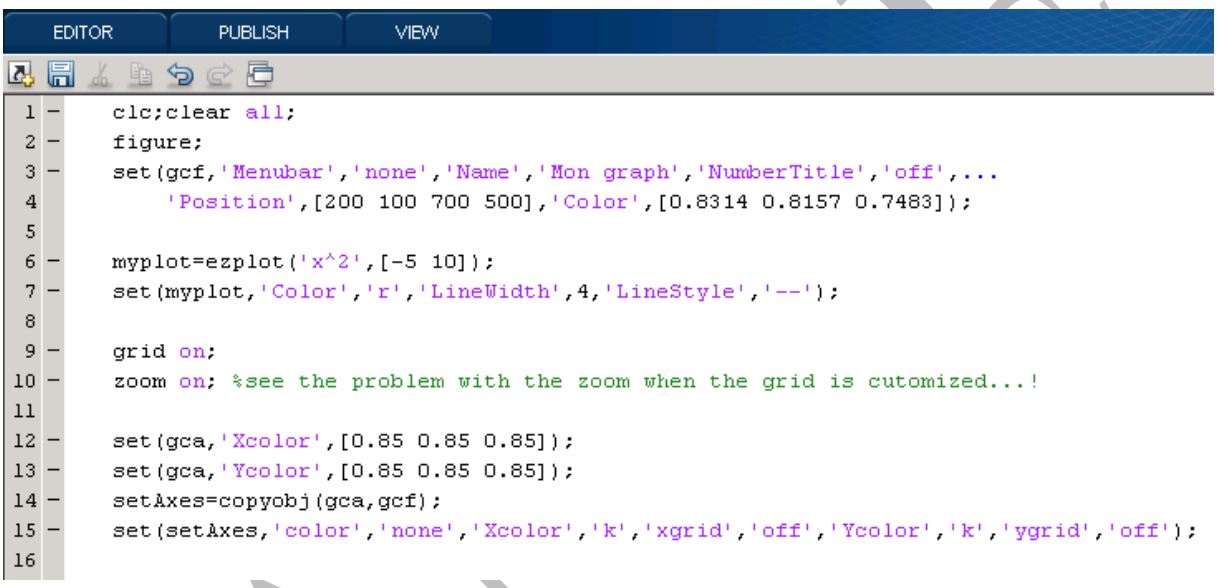
X-Y plots

OK now we begin a big chapter important for students but also for professional practitioners of MATLAB.

The purpose of this chapter is not only to see how to make X-Y but also to give a lot of tips and tricks relatively to the numerous questions of people following my trainings.

Analytical based plot

We start with a very academic example first where many student know the expression of a function a have to simple plot it by typing its expression. You can do this quite easily with the function **ezplot( )**:

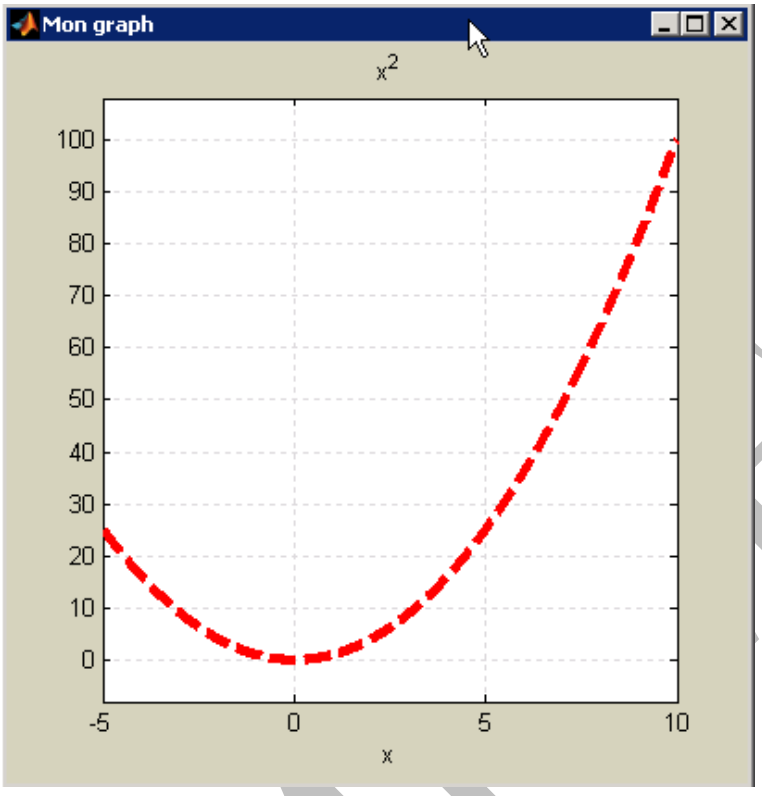



```
1 -    clc;clear all;
2 -    figure;
3 -    set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4          'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);
5
6 -    myplot=ezplot('x^2',[-5 10]);
7 -    set(myplot,'Color','r','LineWidth',4,'LineStyle','--');
8
9 -    grid on;
10 -   zoom on; %see the problem with the zoom when the grid is cutomized...!
11
12 -   set(gca,'Xcolor',[0.85 0.85 0.85]);
13 -   set(gca,'Ycolor',[0.85 0.85 0.85]);
14 -   setAxes=copyobj(gca,gcf);
15 -   set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
16
```

Depending on screen resolution zoom to see the dashlines!!!

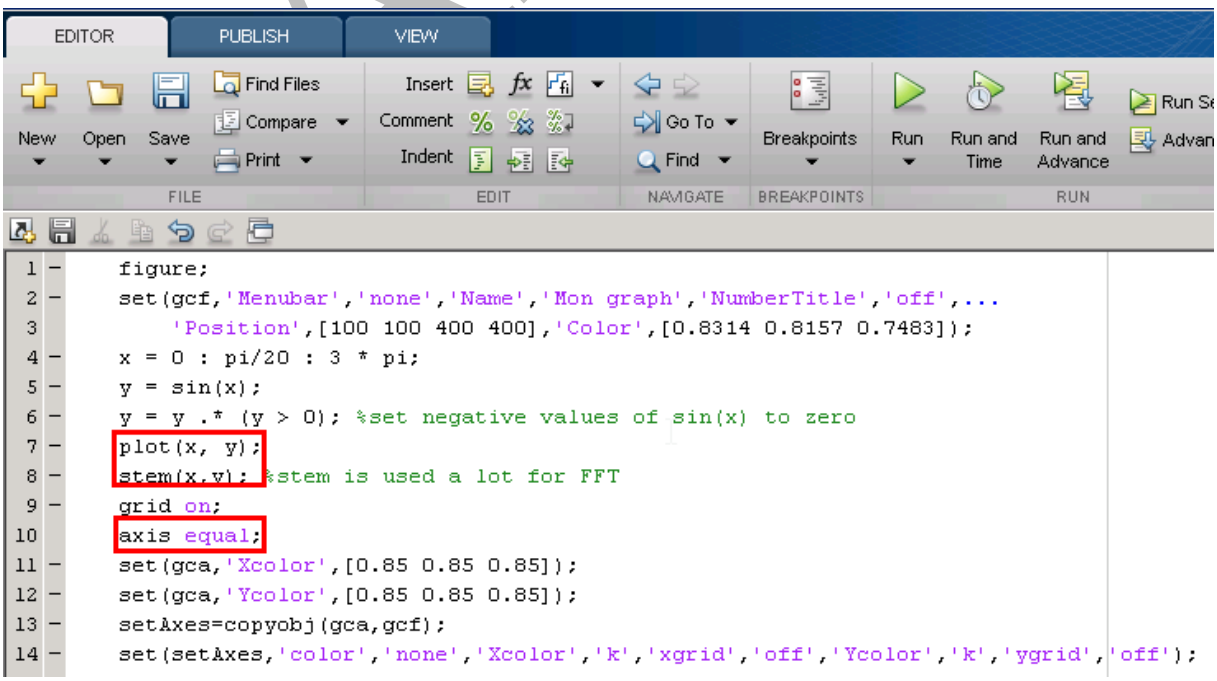
SAMPLE

The result will be the following (see the title that is added automatically):

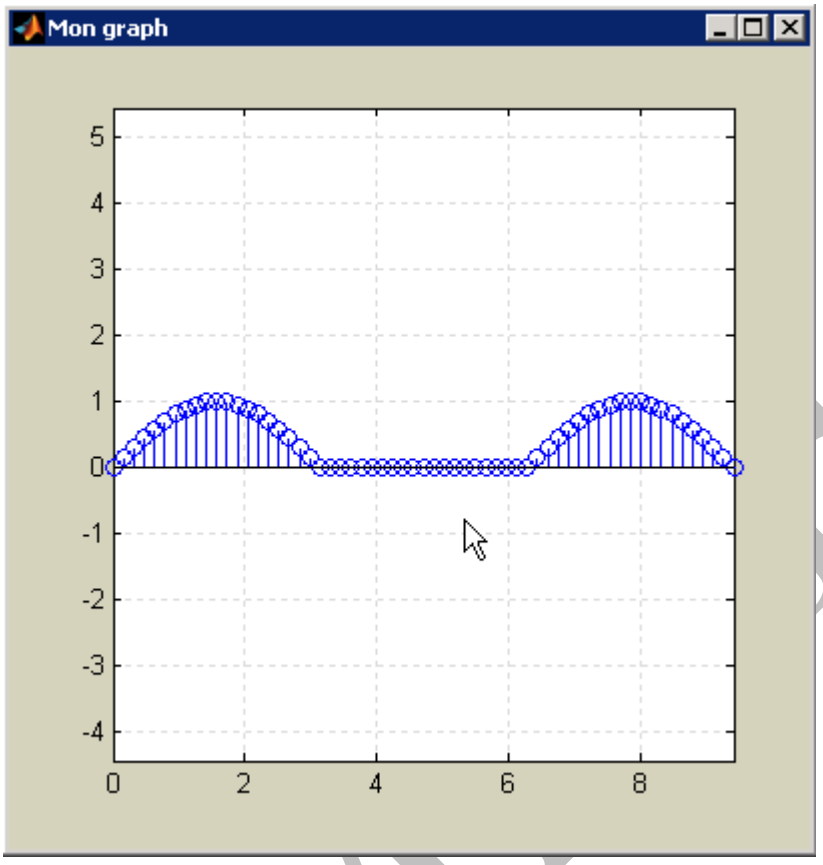


Vector based plot with lines and stems

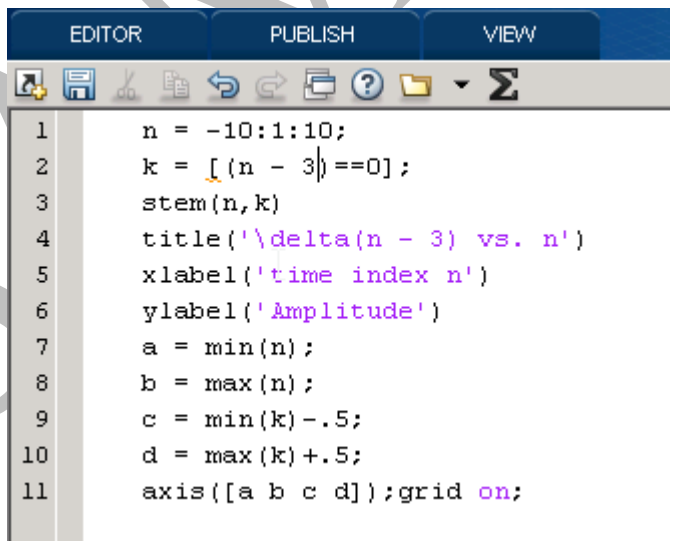
Sometimes and especially in signal processing you can't use analytical expression to plot something. This is what I call "vector based plots". We can make a typical example of this with a simple trigonometric function, a threshold filter condition and using the **stem( )** plot function that is very common is signal processing and see how the chart will looks like by adding or not the line **axis equal**:



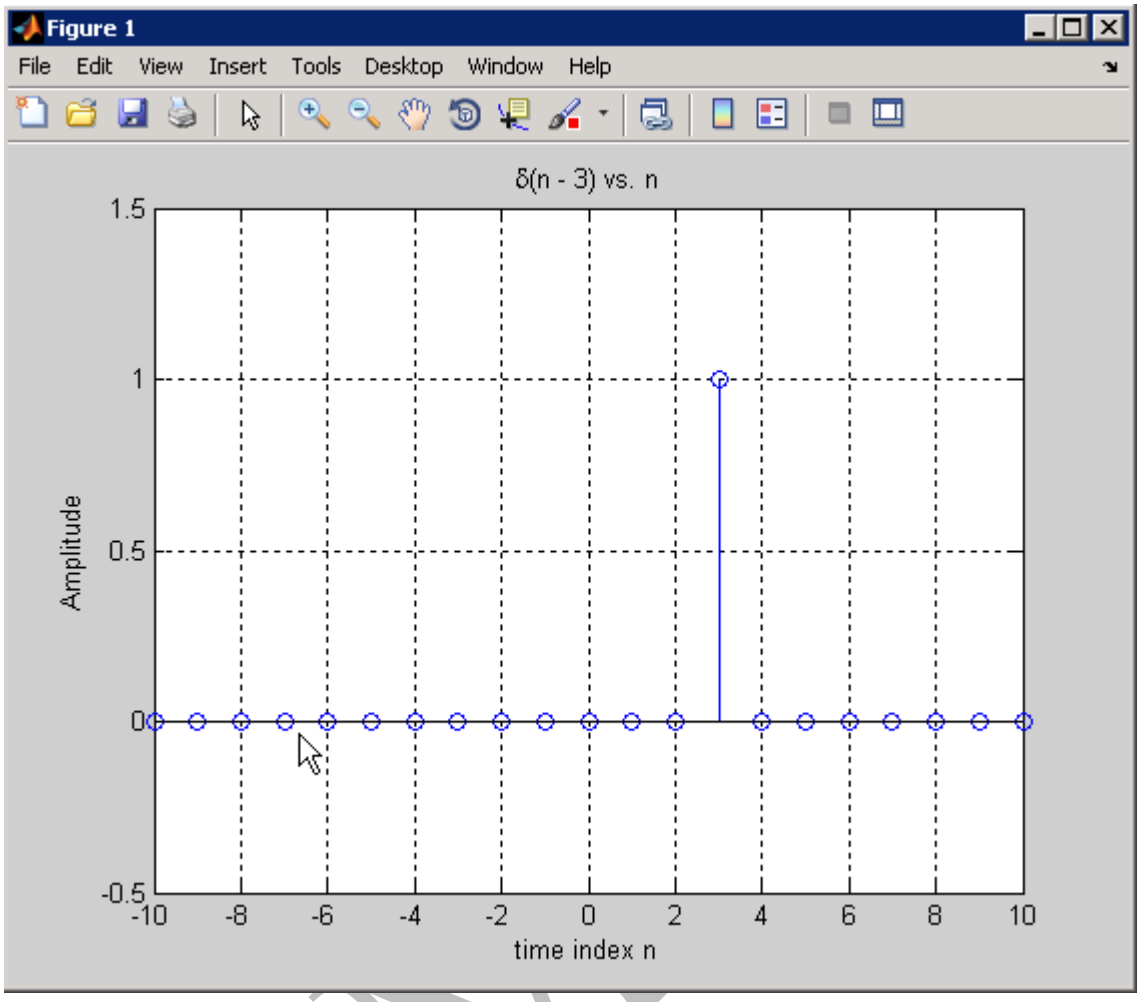
The result will be:



Or a more concrete example:



That gives and Dirac delta on $n = 3$:


Figure 1
File Edit View Insert Tools Desktop Window Help
δ(n - 3) vs. n
Amplitude
time index n
1.5
1
0.5
0
-0.5
-10 -8 -6 -4 -2 0 2 4 6 8 10

SAM

Animated vector based plot

A lot of schools and some financial institutions ask me sometimes how to animate a 2D plots (in financial institutions they like this to animate forecasts...).

Let us see a first method (default native poor one...) using the command **comet( )** and solving at the same time the fact that the plot disappear one the animation is finished:

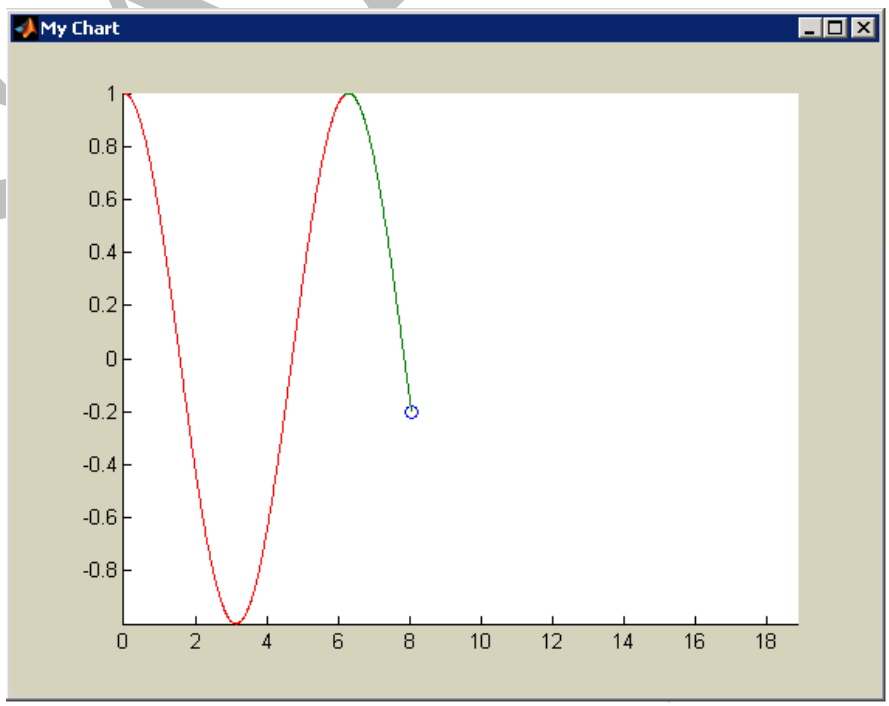
```
clc;clear all;
figure;
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);

grid on;
t=0:0.01:6*pi;
x=t;y=cos(t);
comet(x,y); %the problem is that the line disappears at the end...

%then we need after a normal plot
plot(x,y,'Color','r')
xlim([0 19]); %otherwise it doesn't correspond to comet axis... (stupid behaviour!)
ylabel('y');
xlabel('t');
grid on;
title('Animated 2D plot: cos(t)','fontweigh','bold','fontsize',16);

set(gca,'Xcolor',[0.85 0.85 0.85]);
set(gca,'Ycolor',[0.85 0.85 0.85]);
setAxes=copyobj(gca,gcf);
set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

The result will looks like:



To control speed you can edit the code of the *comet* function by adding a *pause*(2) for example at the right place but then it will work only on your computer. The best way is the following:

Second method using a loop with the functions **pause( )** and **plot( )** to control the speed of the animation:

EDITOR PUBLISH VIEW

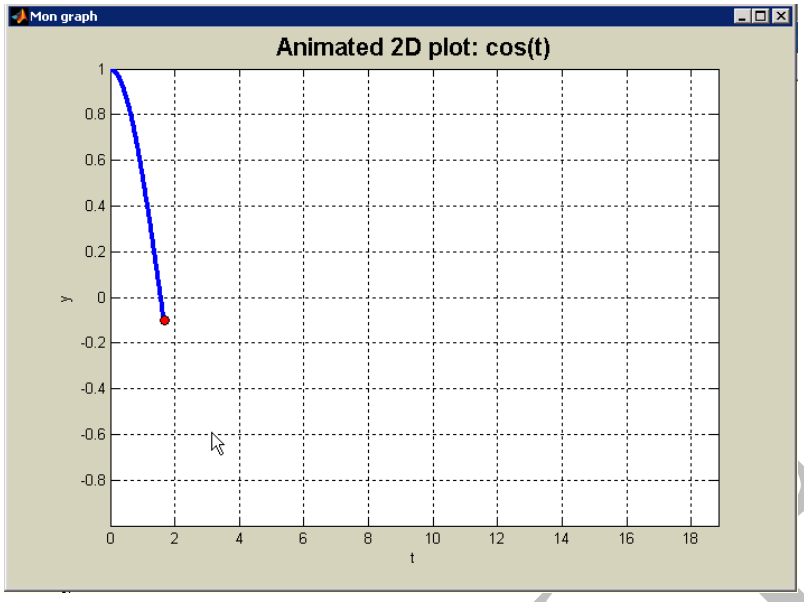
```
clc;clear all;
figure;
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);

t=0:0.01:6*pi;
x=t;y=cos(t);

for k=1:1:length(t)
    pause(0.01); %you can now control speed :-)
    hold off;
    %we draw the line
    plot(x(1:k),y(1:k),'LineWidth',3,'LineStyle','-');
    %and a red point
    hold on;
    plot(x(k),y(k),'ko','markerFaceColor','r');
    axis([min(t) max(t) min(y) max(y)]);
    grid on;
    ylabel('y');
    xlabel('t');
    title('Animated 2D plot: cos(t)','fontweigh','bold','fontsize',16);
end

set(gca,'Xcolor',[0.85 0.85 0.85]);
set(gca,'Ycolor',[0.85 0.85 0.85]);
setAxes=copyobj(gca,gcf);
set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

The result will looks like this:



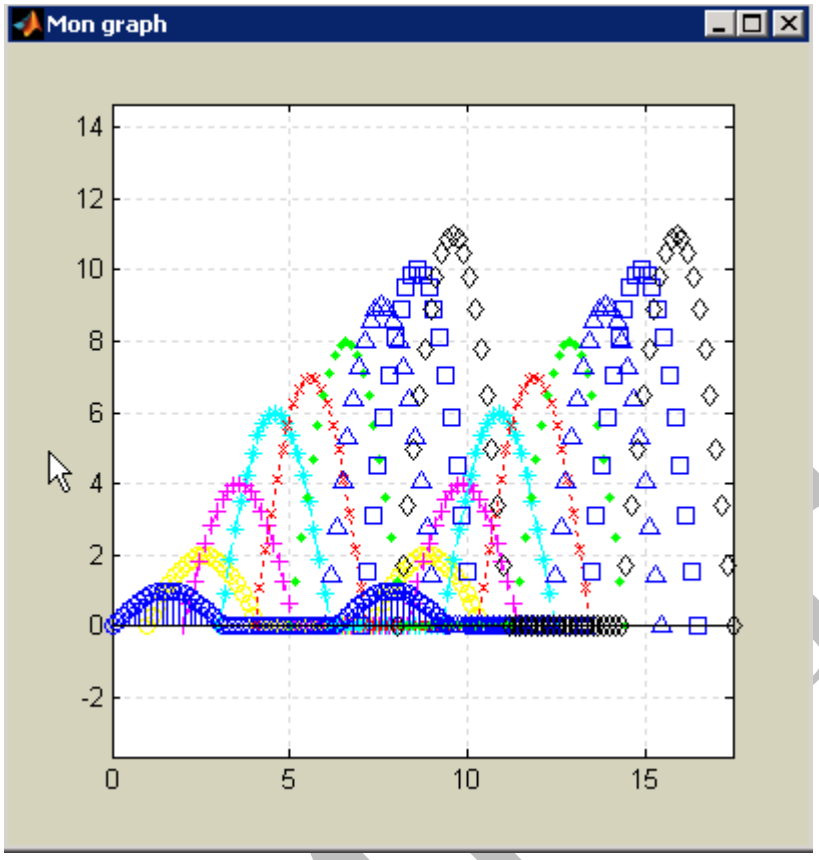
Vector based plot with various lines, colors and points

Depending on the subject you are studying you will need to have multiple lines plots into only one plot area. Then it is important to have an overview of some possible line plots styles.

Here we plot a vector based function but now using the function **plot( )** and we added also a **stem( )** for fun:

EDITOR PUBLISH VIEW

```
1  figure;
2  title('Vector based plot');
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Position',[100 100 400 400],'Color',[0.8314 0.8157 0.7483]);
5  x = 0 : pi/20 : 3 * pi;
6  y = sin(x);
7  y = y .* (y > 0); %set negative values of sin(x) to zero
8  plot(x, y);
9  hold on;
10 plot(x+1, 2*y,'-yo');
11 plot(x+2, 4*y,'-.m+');
12 plot(x+3, 6*y,'--c*');
13 plot(x+4, 7*y,':rx');
14 plot(x+5, 8*y,'.g');
15 plot(x+6, 9*y,'^b');
16 plot(x+7, 10*y,'sb');
17 plot(x+8, 11*y,'kd');
18 stem(x,y); %stem is used a lot for FFT
19 grid on;
20 axis equal;
21 set(gca,'Xcolor',[0.85 0.85 0.85]);
22 set(gca,'Ycolor',[0.85 0.85 0.85]);
23 setAxes=copyobj(gca,gcf);
24 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

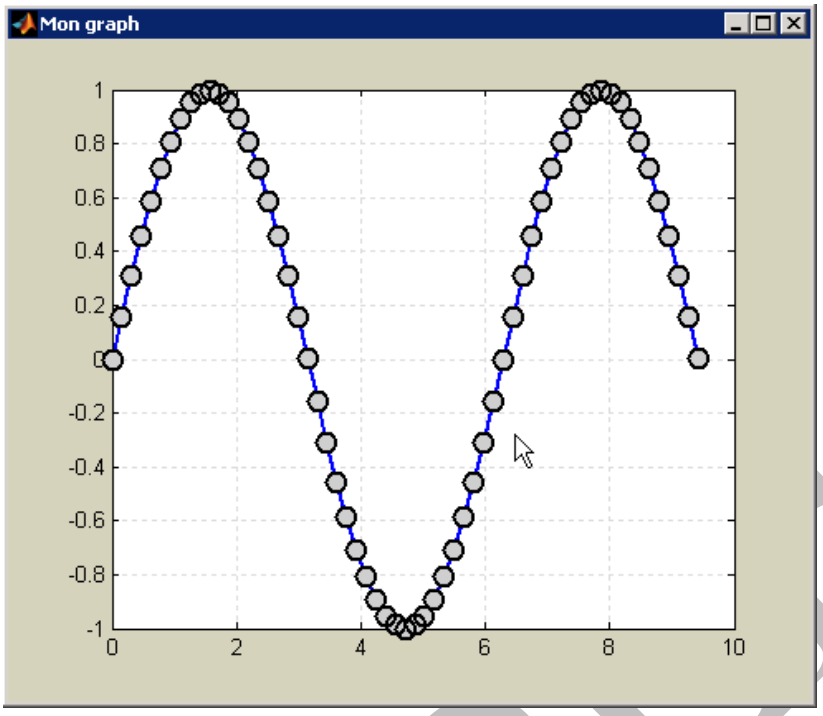


And for line width and some other extras we play with the **plot( )** function arguments:

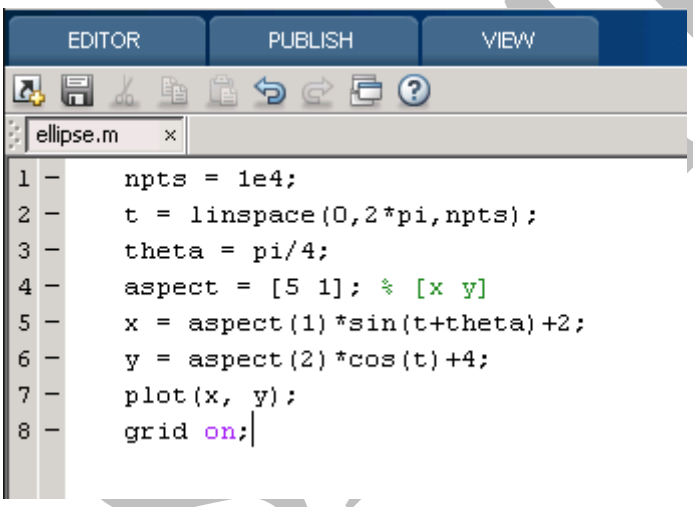
EDITOR PUBLISH VIEW

```
1 -    figure;
2 -    set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
3          'Position',[100 100 400 400],'Color',[0.8314 0.8157 0.7483]);
4
5 -    x=0:pi/20:3*pi;
6 -    y=sin(x);
7 -    plot(x,y,'-bo','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','k',...
8          'MarkerFaceColor',[0.8 0.8 0.8])
9
10 -   grid on;
11 -   set(gca,'Xcolor',[0.85 0.85 0.85]);
12 -   set(gca,'Ycolor',[0.85 0.85 0.85]);
13 -   setAxes=copyobj(gca,gcf);
14 -   set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

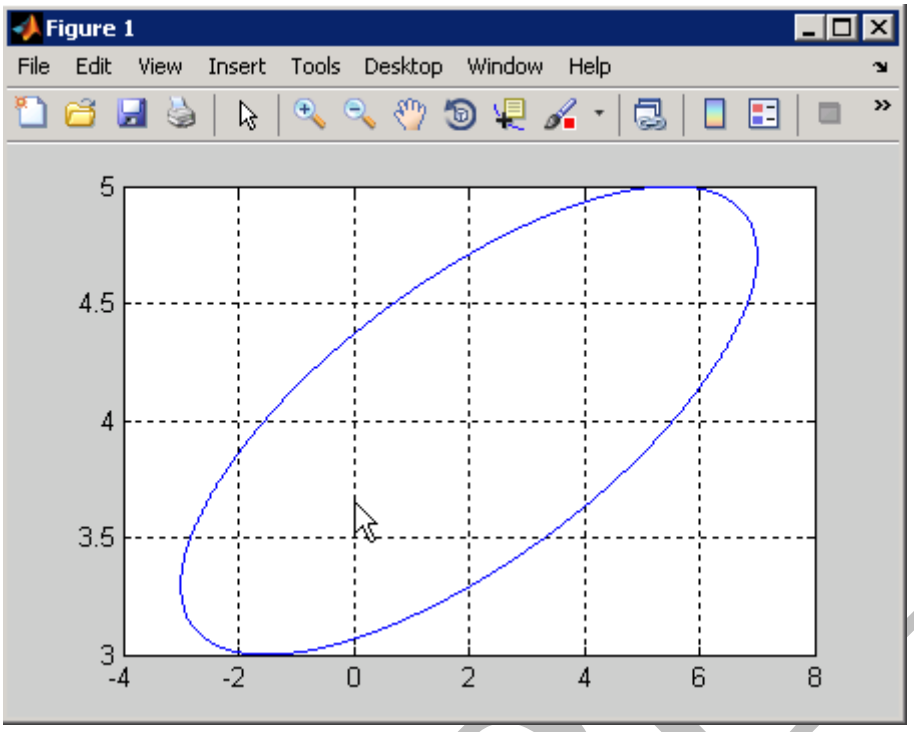
See the result next page.



Another famous example (used a lot in engineering and image processing) is hot to draw and ellipse:



This will give you:



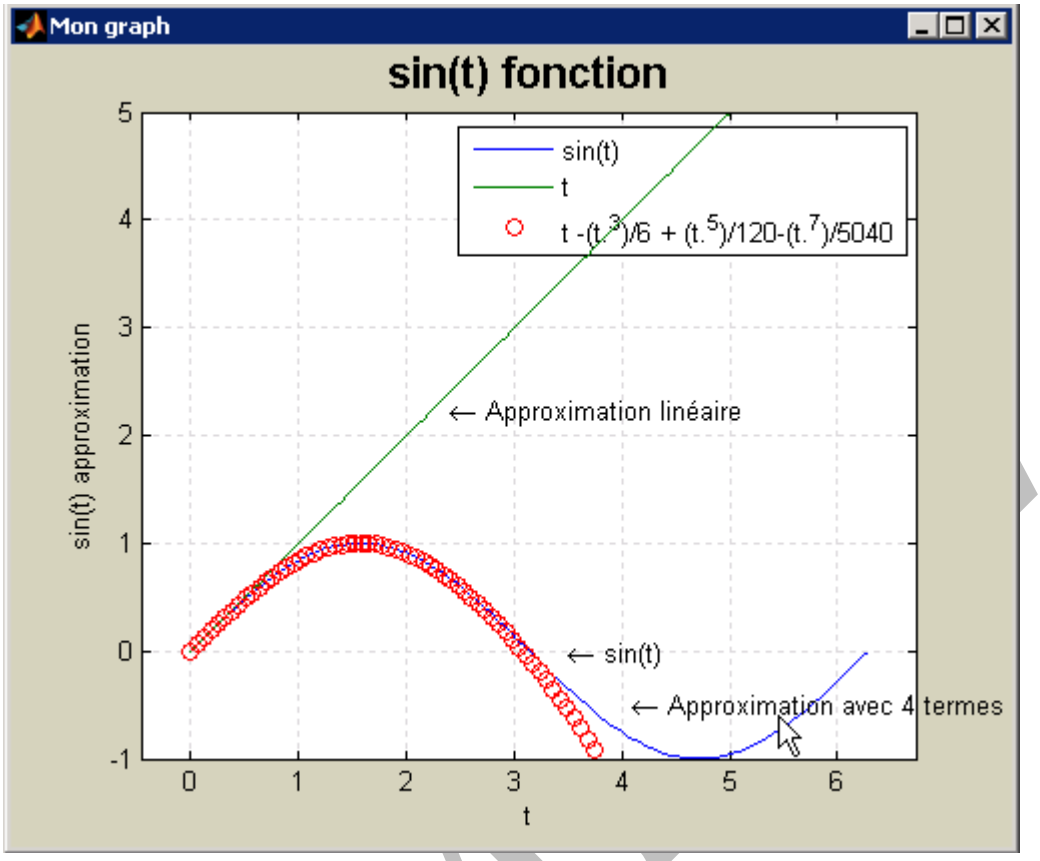
Vector based multiple plot with legends and titles

Instead than typing multiple times plot( ), plot( ), plot( ) etc. you can directly in only one plot( ) function pass multiple plot vectors. At the same time the purpose of this example is to show add axes labels with **xlabel( )** and **ylabel( )**, plot title with **title( )**, plot legends with **legend( )** and static text with **text( )** and dynamic text with **gtext( )** on the plot:

EDITOR PUBLISH VIEW

```
1  clc; clear all
2  figure;
3  title('Vector based plot');
4  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
5      'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6  t = linspace(0, 2*pi, 100);
7  y1 = sin(t);
8  y2 = t;
9  y3 = t -(t.^3)/6 + (t.^5)/120-(t.^7)/5040;
10 plot(t, y1, t, y2, '-', t, y3, 'o');
11 grid on;
12 axis([0 5 -1 5]);
13 xlabel('t');
14 ylabel('sin(t) approximation');
15 title('sin(t) fonction','fontweight','bold','fontsize',16);
16 text(3.5,0, '\leftarrow sin(t)');
17 legend('sin(t)','t','t -(t.^3)/6 + (t.^5)/120-(t.^7)/5040');
18 gtext('\leftarrow Approximation linéaire');
19 gtext('\leftarrow Approximation avec 4 termes');
20 axis equal;
21 set(gca,'Xcolor',[0.85 0.85 0.85]);
22 set(gca,'Ycolor',[0.85 0.85 0.85]);
23 setAxes=copyobj(gca,gcf);
24 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

This will result in:



Thanks to **ginput( )** the texts will be placed where the user has click on the chart.

SAMPLE

Horizontal/Vertical lines method

A very very common question to almost all jobs specializations is how to add a vertical/horizontal line on chart. This can easily be done thanks to the **graph2d.constantline( )** function:

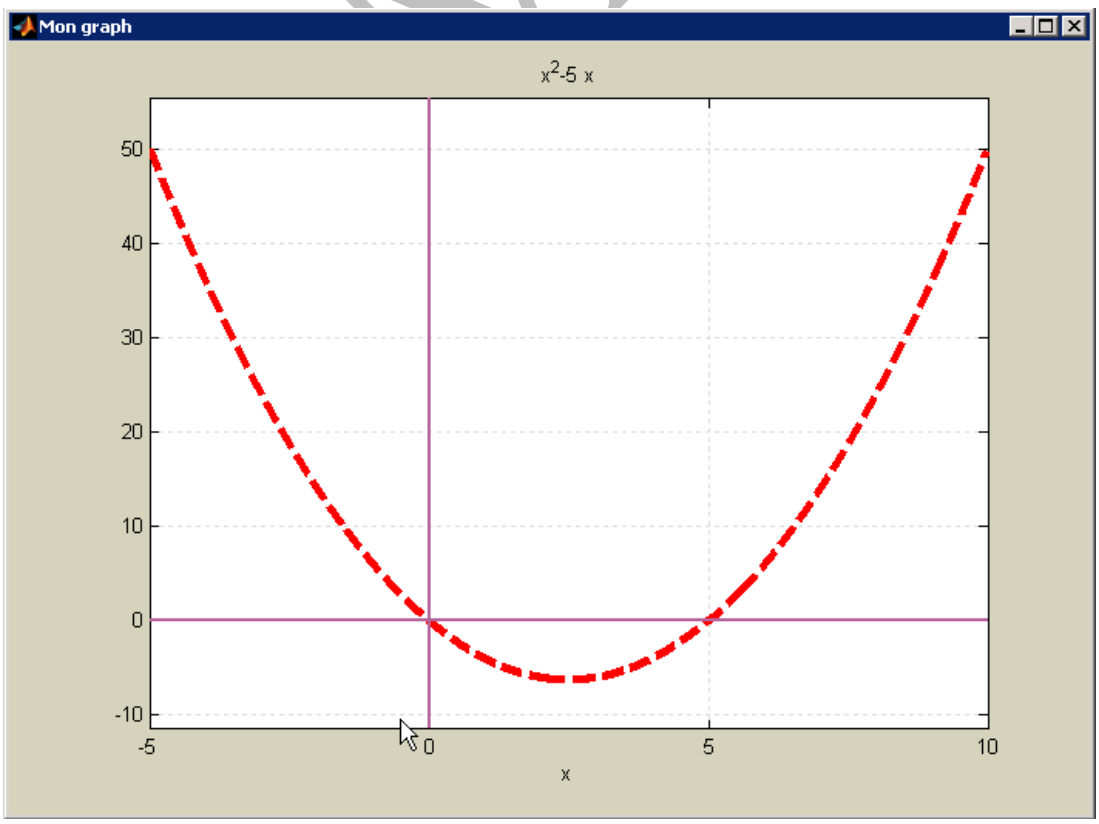
```
clc;clear all;
figure;
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);

myplot=ezplot('x^2-5*x',[-5 10]);
set(myplot,'Color','r','LineWidth',4,'LineStyle','--');
grid on;

%vertical line
hx=graph2d.constantline(0,'Color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hx,'x');
%horizontal line
hy=graph2d.constantline(0,'Color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hy,'y');

set(gca,'Xcolor',[0.85 0.85 0.85]);
set(gca,'Ycolor',[0.85 0.85 0.85]);
setAxes=copyobj(gca,gcf);
set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

The result will look as following:

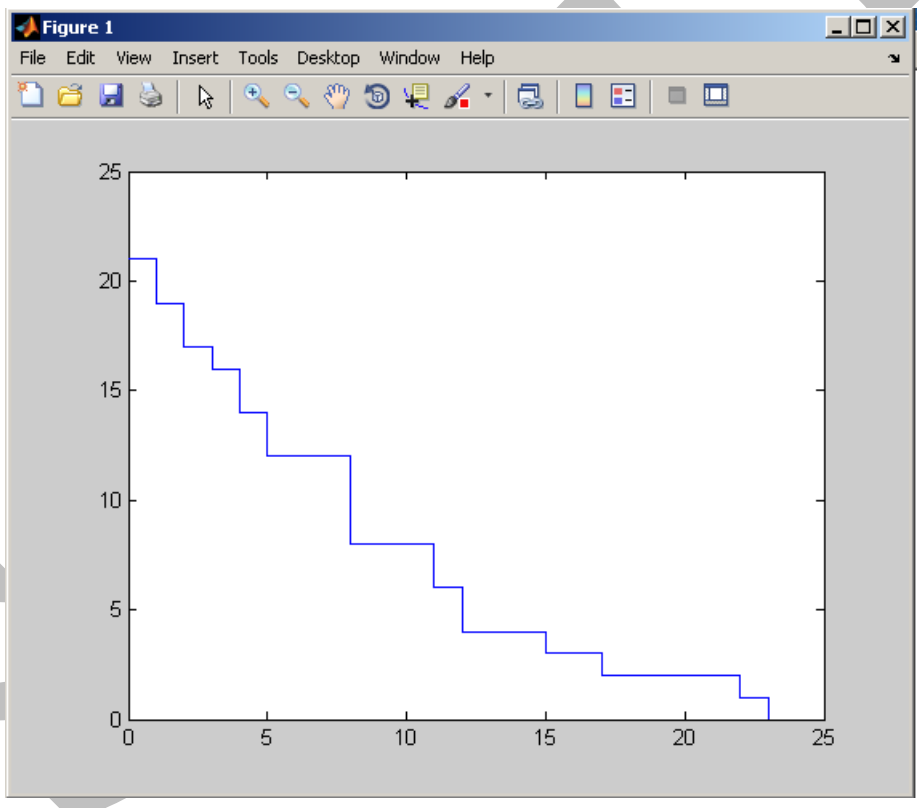


Stairs Plot

Used a lot in guarantee and survival analysis. We will here reproduce the example of the theoretical course during our study of the Kaplan-Meier model:

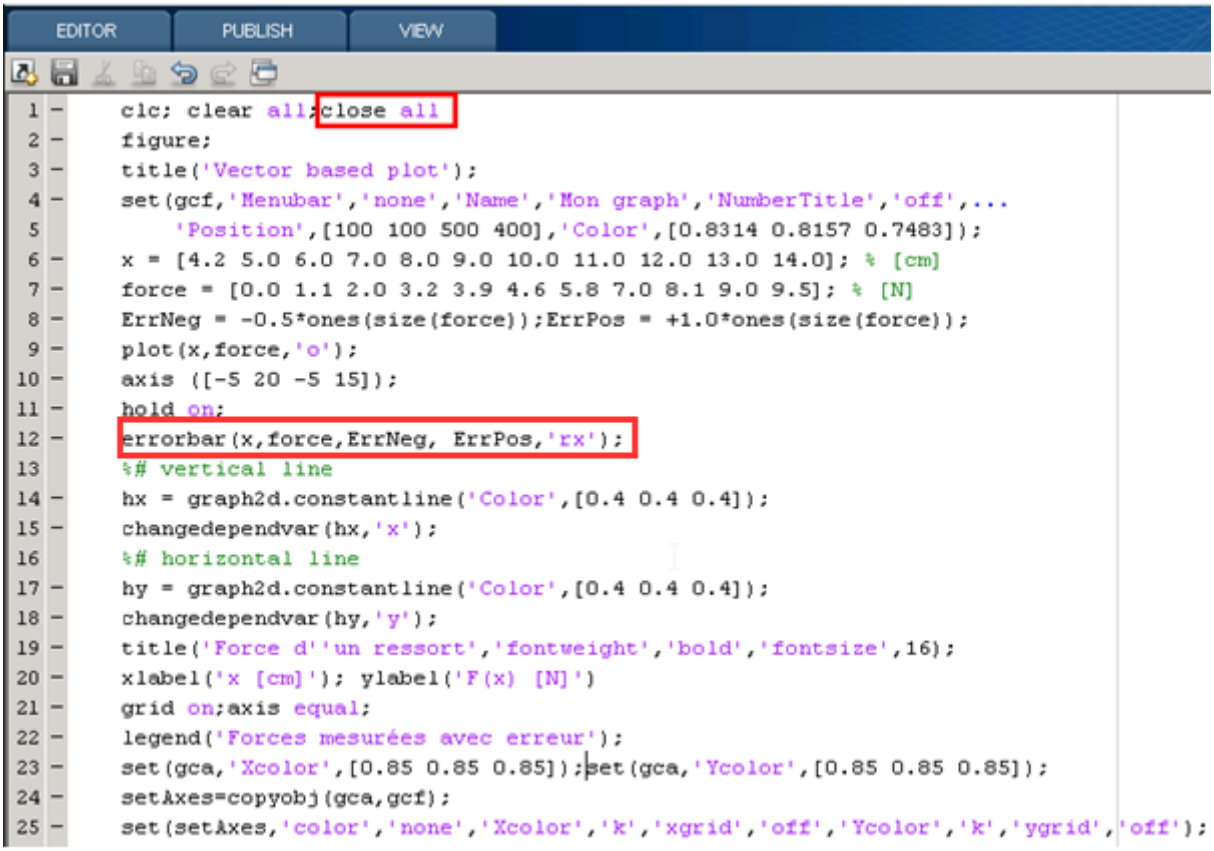
```
MATLAB Command Window
» X = [0 1 2 3 4 5 8 11 12 15 17 22 23];
» Y = [21 19 17 16 14 12 8 6 4 3 2 1 0];
» stairs(X,Y)
»
```

Which gives:



Vector based plot with error bars

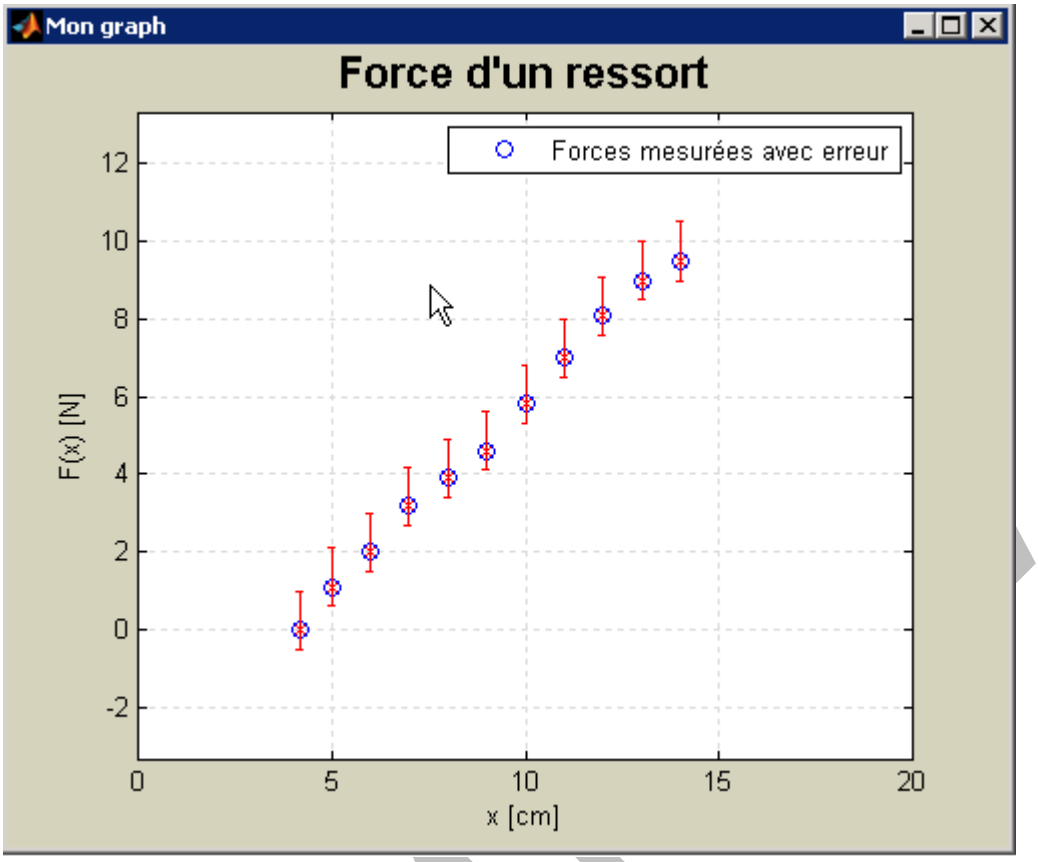
Ok now see how we do a dot plot with error bars in MATLAB using the function **errorbar( )**. At the same time see that at the beginning we close all open charts to be sure that our will be the only one opened in this MATLAB sessions:



```
clc; clear all;close all
figure;
title('Vector based plot');
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
x = [4.2 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0]; % [cm]
force = [0.0 1.1 2.0 3.2 3.9 4.6 5.8 7.0 8.1 9.0 9.5]; % [N]
ErrNeg = -0.5*ones(size(force));ErrPos = +1.0*ones(size(force));
plot(x,force,'o');
axis ([-5 20 -5 15]);
hold on;
errorbar(x,force,ErrNeg, ErrPos,'rx');
%# vertical line
hx = graph2d.constantline('Color',[0.4 0.4 0.4]);
changedependvar(hx,'x');
%# horizontal line
hy = graph2d.constantline('Color',[0.4 0.4 0.4]);
changedependvar(hy,'y');
title('Force d''un ressort','fontweight','bold','fontsize',16);
xlabel('x [cm]'); ylabel('F(x) [N]')
grid on;axis equal;
legend('Forces mesurées avec erreur');
set(gca,'Xcolor',[0.85 0.85 0.85]);set(gca,'Ycolor',[0.85 0.85 0.85]);
setAxes=copyobj(gca,gcf);
set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

See the result on next page.

SAMPLE



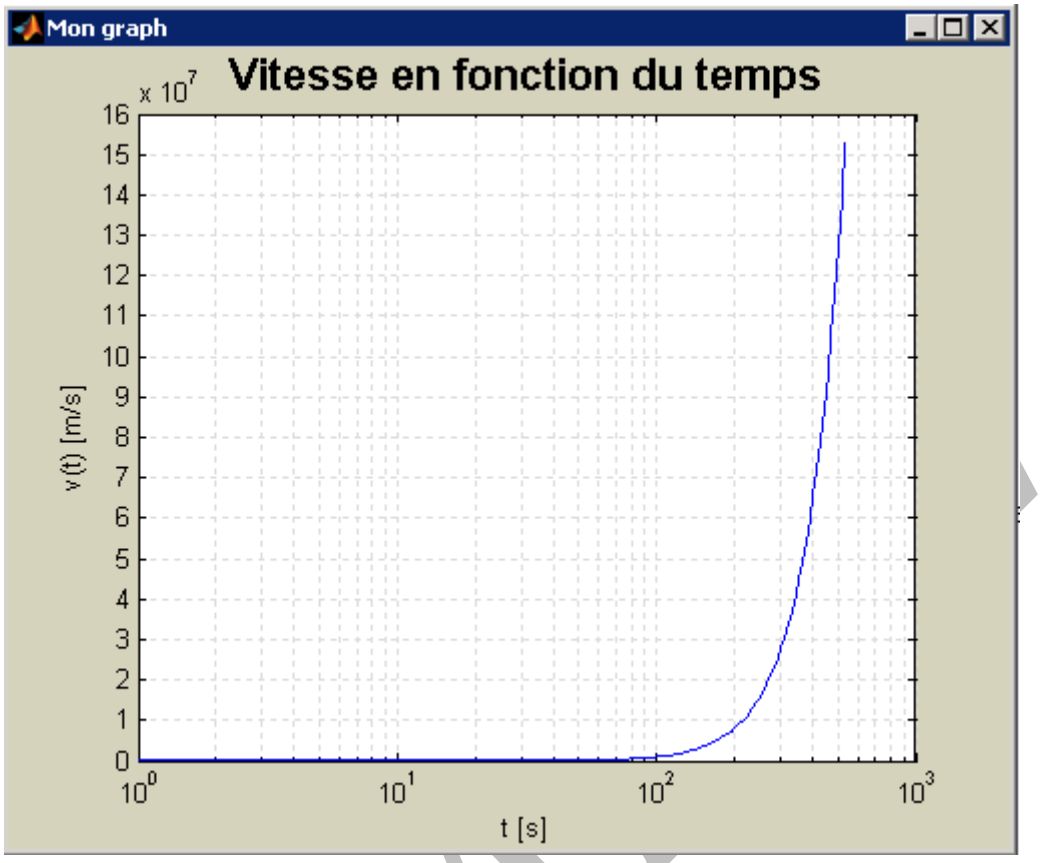
Vector based semi-log X plot with custom tick space

Now see the famous semi-log plots used a lot by physicist and electronic engineers using the the function **semilogx( )** and see how we can control the axes Y tick space at the same time using the **YTick** property of the active chart figure:

EDITOR PUBLISH VIEW

```
1  clc; clear all;close all
2  figure;
3  title('Vector based plot');
4  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
5      'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6  t = linspace(0, 2*pi, 200);
7  x = exp(t);
8  y = 100 + exp(3*t);
9  semilogx(x, y);
10 title('Vitesse en fonction du temps','fontweight','bold','fontsize',16);
11 xlabel('t [s]'); ylabel('v(t) [m/s]')
12 grid on;
13 set(gca,'YTick',[0:10000000:200000000])
14 %axis equal; must be removed when using log scale!
15 set(gca,'Xcolor',[0.85 0.85 0.85]);set(gca,'Ycolor',[0.85 0.85 0.85]);
16 setAxes=copyobj(gca,gcf);
17 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
18
```

This is how the result looks like:



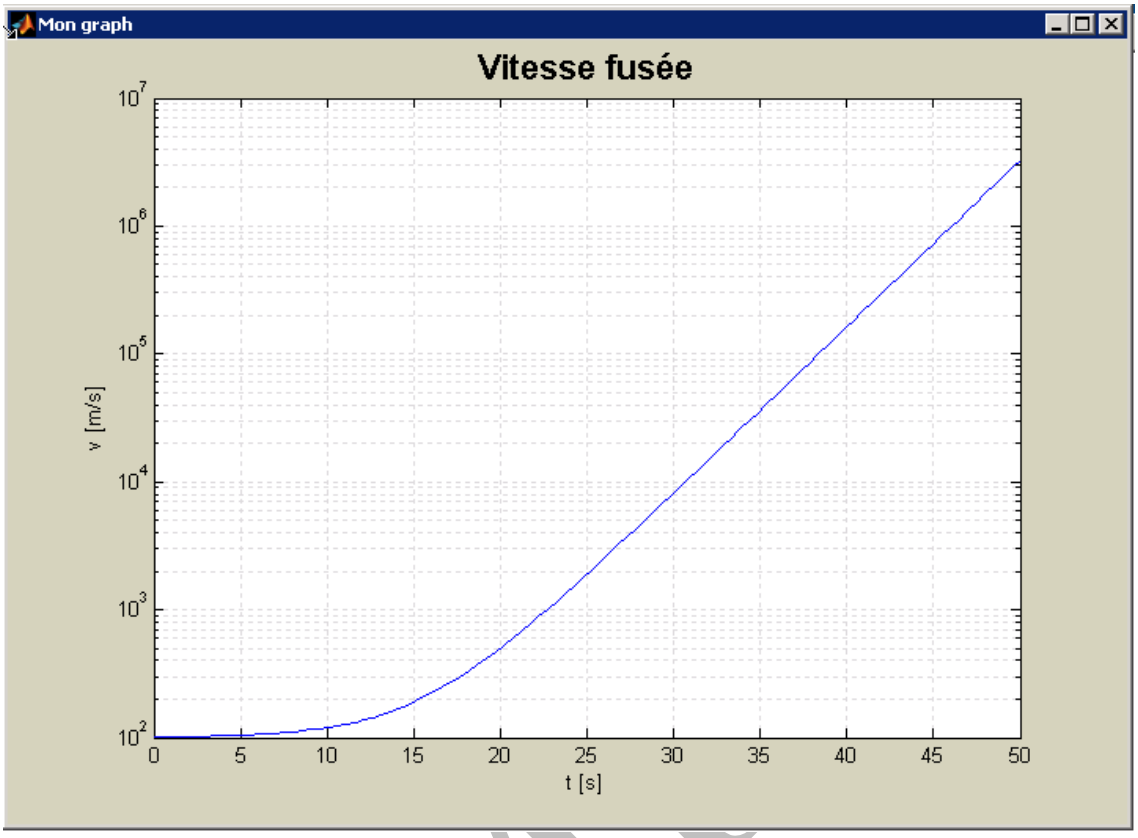
Vector based semi-log Y plot

And in the same idea but using with the Y axes using the **semilogy( )** function:

```
EDITOR  PUBLISH  VIEW

1  clc;clear all;close all;
2  figure;
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Position',[200 -50 800 800],'Color',[0.8314 0.8157 0.7483]);
5
6  t=linspace(0,50,100);
7  v=100+exp(0.3*t);
8  semilogy(t,v);
9
10 xlabel('t [s]');
11 ylabel('v [m/s]');
12 title('Vitesse fusée','fontweight','bold','fontsize',16);
13
14 grid on;
15 set(gca,'Xcolor',[0.85 0.85 0.85]);
16 set(gca,'Ycolor',[0.85 0.85 0.85]);
17 setAxes=copyobj(gca,gcf);
18 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
19
20 datacursormode on;
```

This is how the result looks like:



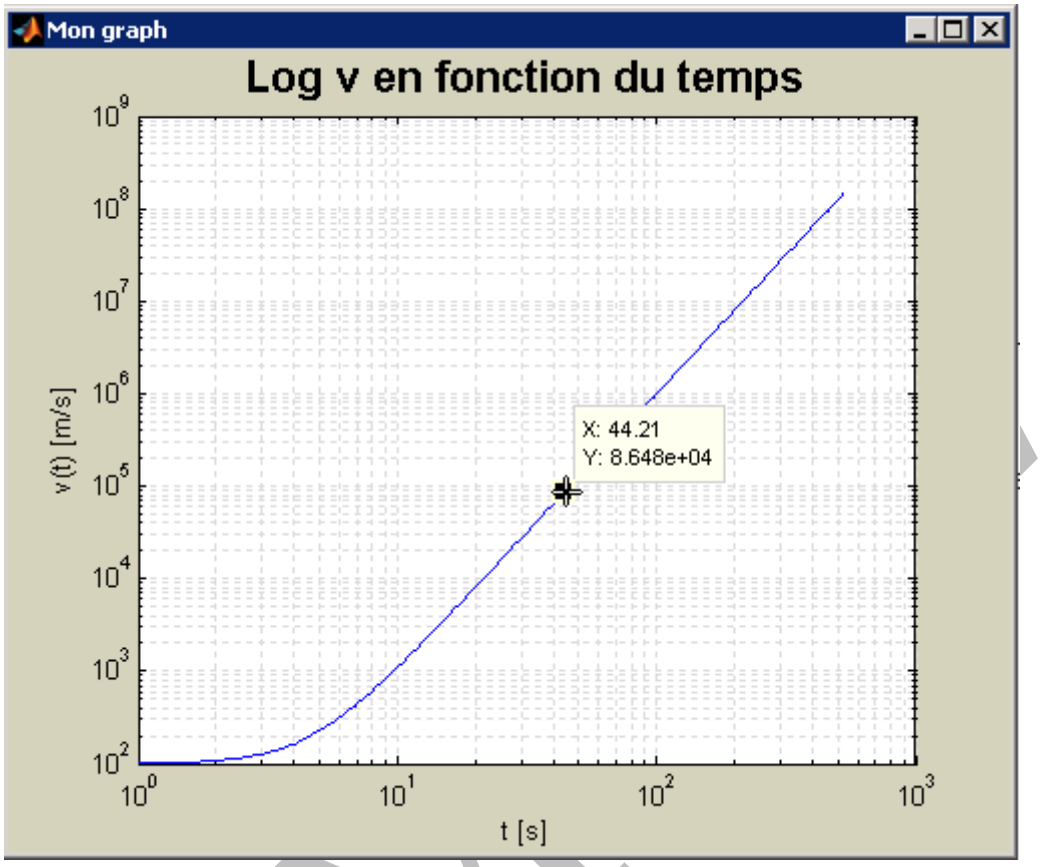
Vector based log-log plot with data cursor tip

This case is rare... using a log-log plot thanks to the function **loglog( )**. What is instead interesting in this chart is that we show here for the first time that you can give to your user/customer the possibility to play with the data cursor by using the command **datacursormode on**:

EDITOR PUBLISH VIEW

```
1  clc; clear all;close all
2  figure;
3  title('Vector based plot');
4  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
5      'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6  t = linspace(0, 2*pi, 200);
7  x = exp(t);
8  y = 100+ exp(3*t);
9  loglog(x, y);
10 title('Log v en fonction du temps','fontweight','bold','fontsize',16);
11 xlabel('t [s]'); ylabel('v(t) [m/s]')
12 grid on;
13 %axis equal; must be removed when using log scale!
14 set(gca,'Xcolor',[0.85 0.85 0.85]);set(gca,'Ycolor',[0.85 0.85 0.85]);
15 setAxes=copyobj(gca,gcf);
16 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
17
18 datacursormode on
```

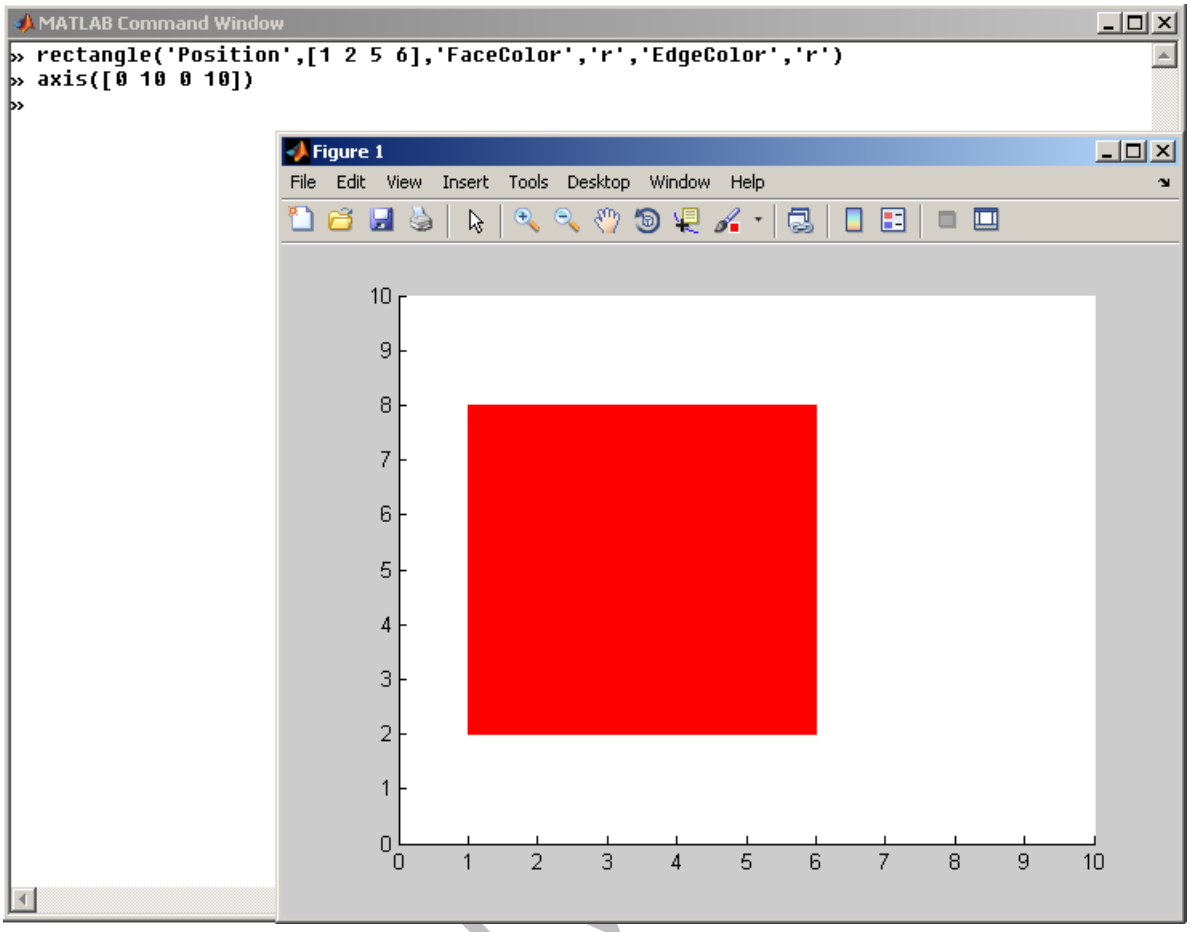
Now with the data cursor mode even if there are no toolbars anymore the user/customer can still click on any point of the chart to get some data tips on the screen:



SAMPLE

Simple shapes (circles, rectangles)

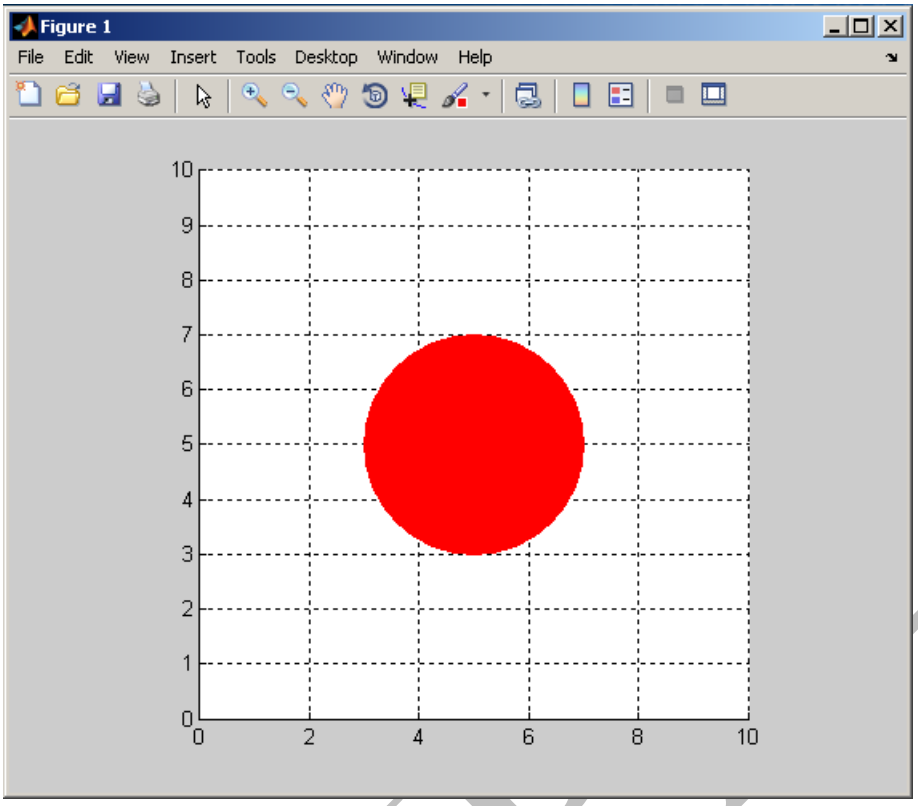
Create a rectangle is quite simple in a MATLAB plot:



For a circle it is less obvious:

```
radius=2;
centerX=5;centerY=5;
rectangle('Position',[centerX-radius,centerY-radius,radius*2,radius*2],...
    'Curvature',[1,1], 'FaceColor','r','EdgeColor','r')
axis([0 10 0 10])
axis square;
grid on;
```

That gives:

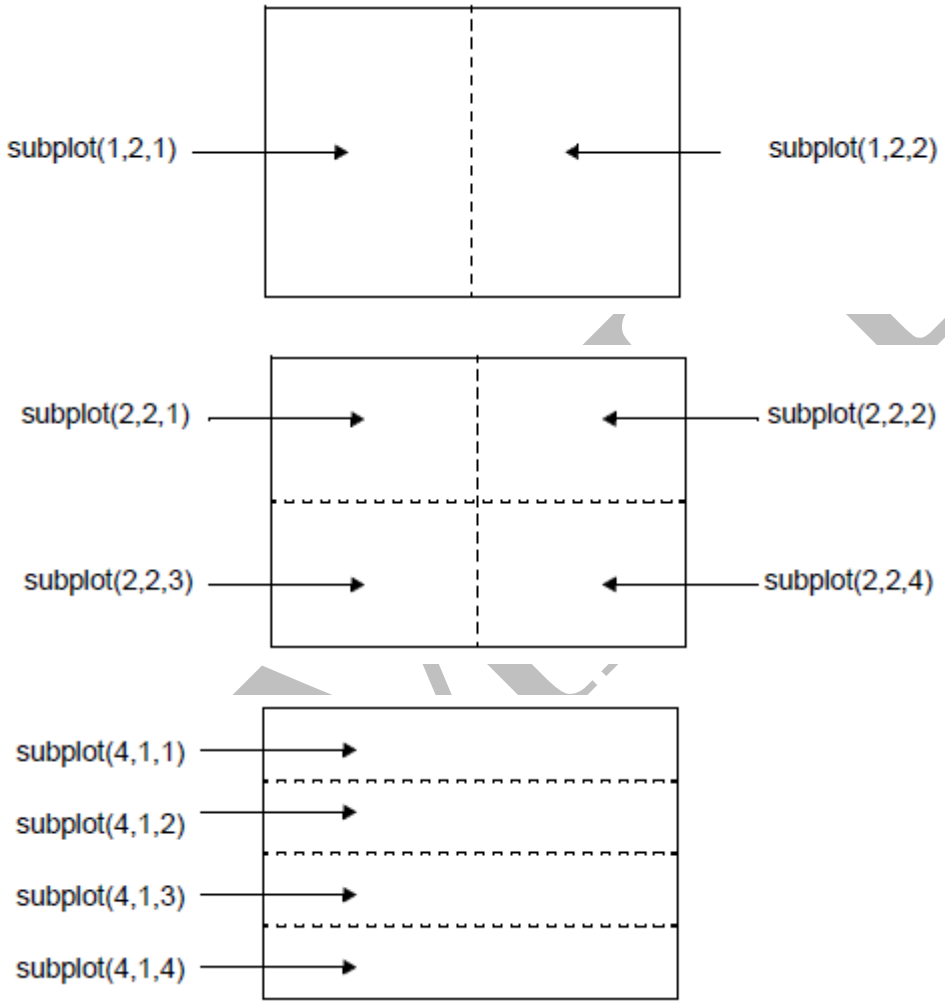

Figure 1
File Edit View Insert Tools Desktop Window Help
10
9
8
7
6
5
4
3
2
1
0
0 2 4 6 8 10

SAMPLE

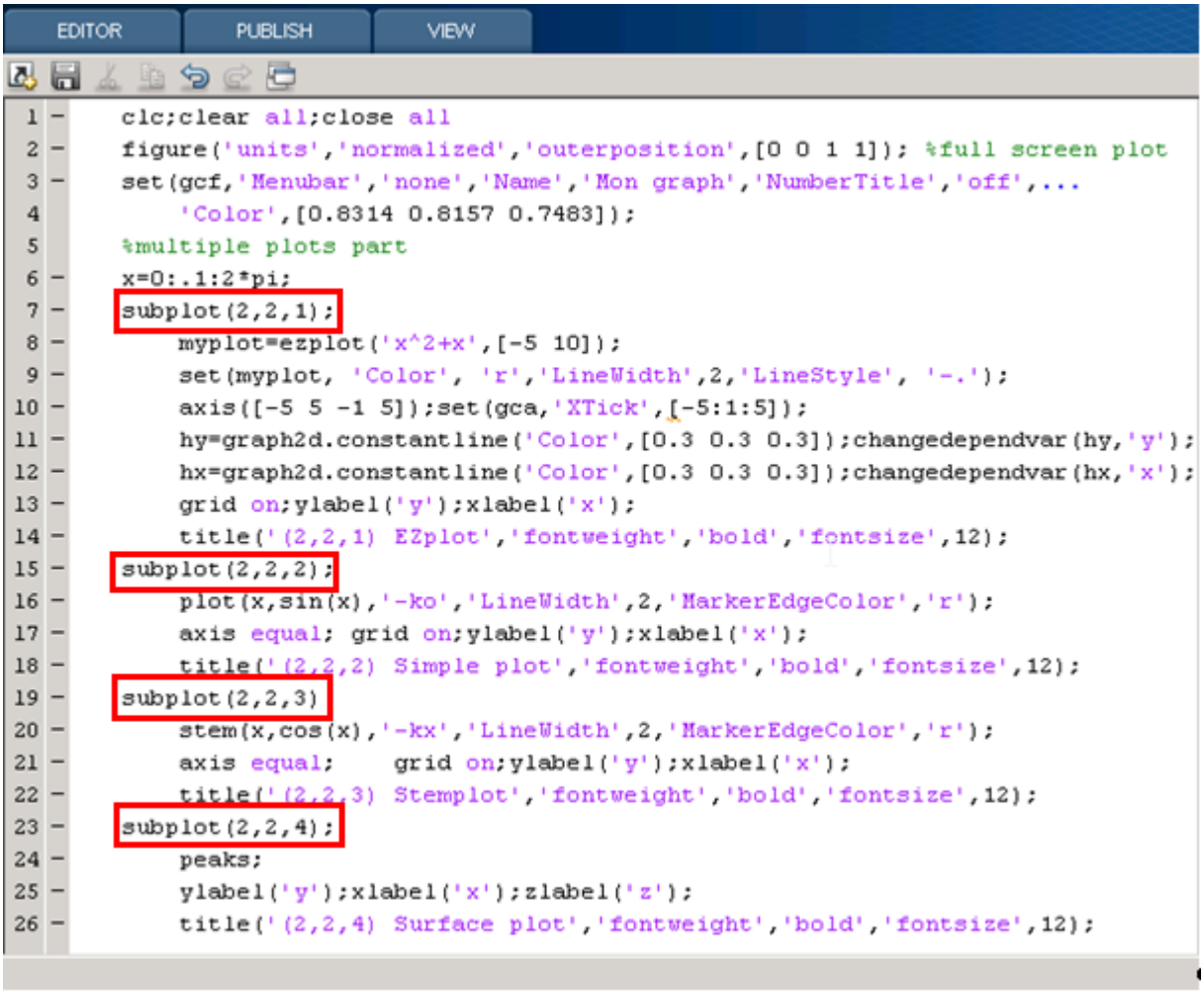
Multiple plots (subplots, lattices)

Subplots

Having multiple windows to show multiple charts can be very boring. You have the possibility in MATLAB as in R to create multiple plots window using the function **subplot(rows, columns, cell number)**. The main ideas is the following:



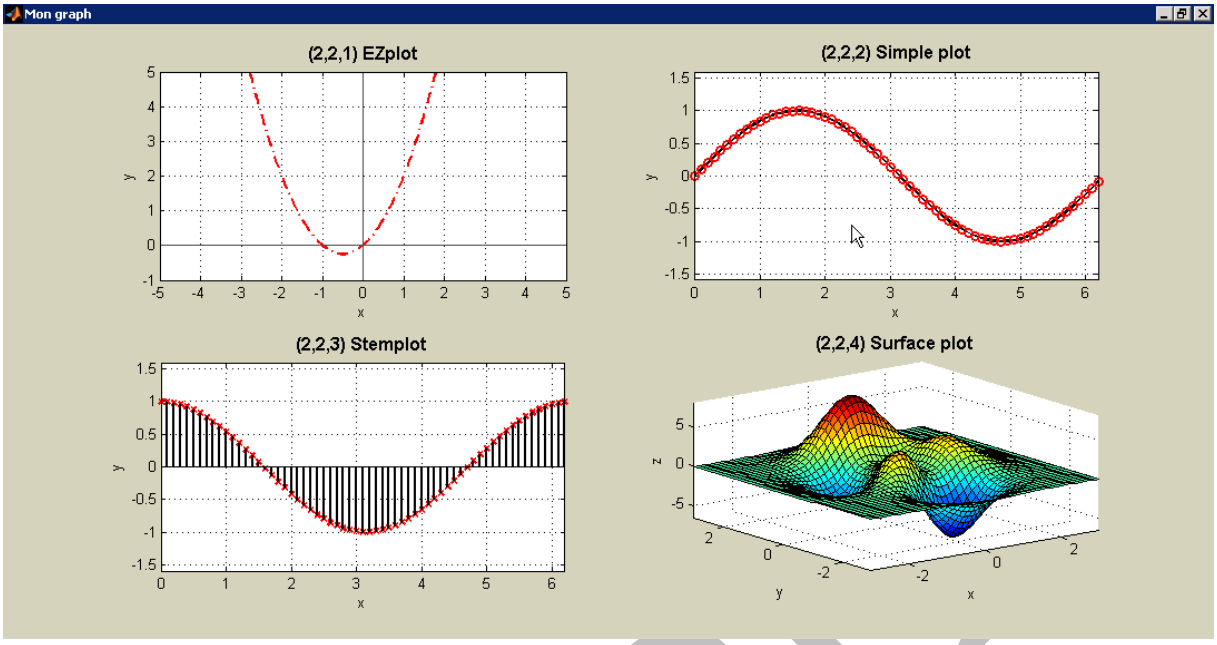
Here is an example:




```
clc;clear all;close all
figure('units','normalized','outerposition',[0 0 1 1]); %full screen plot
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);
%multiple plots part
x=0:.1:2*pi;
subplot(2,2,1);
    myplot=ezplot('x^2+x',[-5 10]);
    set(myplot, 'Color', 'r','LineWidth',2,'LineStyle', '-.');
    axis([-5 5 -1 5]);set(gca,'XTick',[-5:1:5]);
    hy=graph2d.constantline('Color',[0.3 0.3 0.3]);changedependvar(hy,'y');
    hx=graph2d.constantline('Color',[0.3 0.3 0.3]);changedependvar(hx,'x');
    grid on;ylabel('y');xlabel('x');
    title('(2,2,1) EZplot','fontweight','bold','fontsize',12);
subplot(2,2,2);
    plot(x,sin(x),'-ko','LineWidth',2,'MarkerEdgeColor','r');
    axis equal; grid on;ylabel('y');xlabel('x');
    title('(2,2,2) Simple plot','fontweight','bold','fontsize',12);
subplot(2,2,3)
    stem(x,cos(x),'-kx','LineWidth',2,'MarkerEdgeColor','r');
    axis equal;    grid on;ylabel('y');xlabel('x');
    title('(2,2,3) Stemplot','fontweight','bold','fontsize',12);
subplot(2,2,4);
    peaks;
    ylabel('y');xlabel('x');zlabel('z');
    title('(2,2,4) Surface plot','fontweight','bold','fontsize',12);
```

SAM

This will result to:



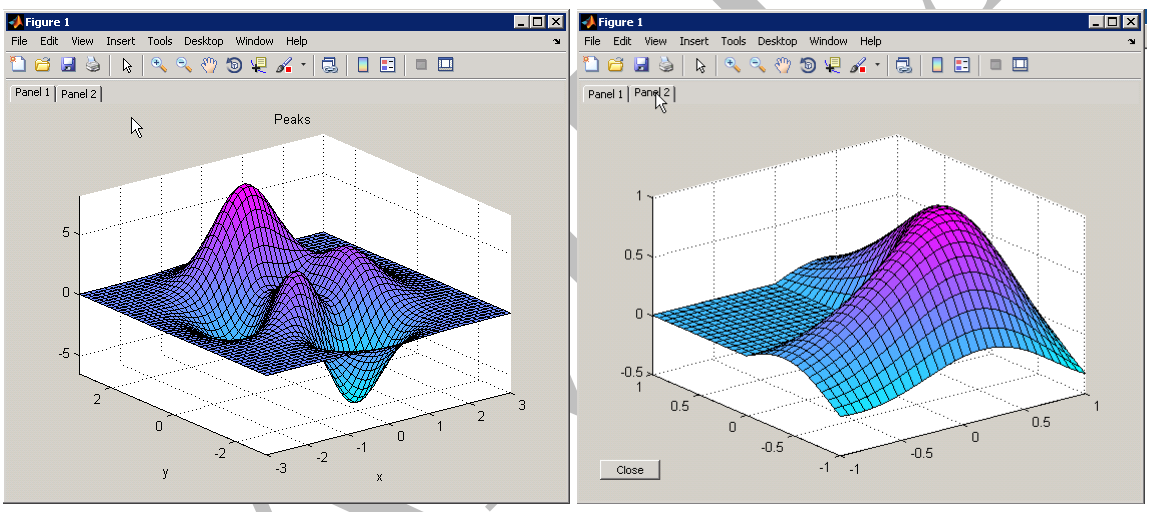
SAMPLE

Tab plots

The commands **uitabgroup** and the function **uitab( )** show below have been officially released with MATLAB 2014a but were still here long before (hidden undocumented commands). So here is a small example:

```
1    hTabGroup = uitabgroup; drawnow;
2    tab1 = uitab(hTabGroup, 'title','Panel 1');
3    a = axes('parent', tab1); peaks;
4    tab2 = uitab(hTabGroup, 'title','Panel 2');
5    a = axes('parent', tab2); membrane;
6    uicontrol(tab2, 'String','Close', 'Callback','close(gcbf)');
```

That gives:

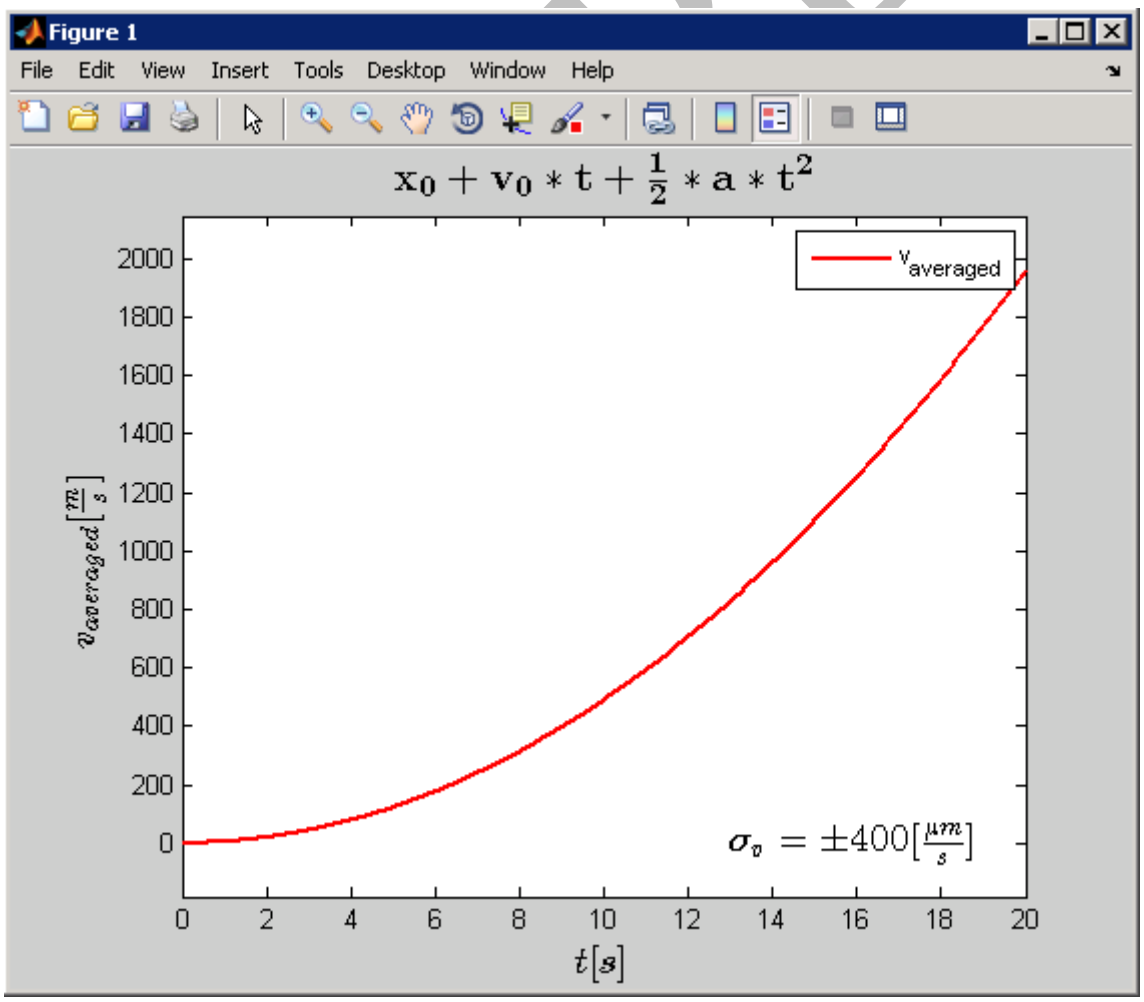


Greeks letters in legends and titles (LaTeX style)

A very important option in any mathematical software is the ability to use LaTeX to write equations or mathematical symbols for different outputs. This is also possible with MATLAB by using the "**interpreter**". You can see 5 examples just below:

```
clear all;close all;clc
x0=0;v0=0;a=9.81;
myplot=ezplot(sprintf('%f+%f*t+0.5*%f*t^2',x0,v0,a),[0 20]);
set(myplot,'color','r','LineWidth',2,'LineStyle','-');
ylabel('$v_{averaged} [\frac{m}{s}]$','interpreter','latex','fontsize',14);
xlabel('$t [s]$','interpreter','latex','fontsize',14);
legend('v_{averaged}','interpreter','latex','fontsize',14);
text(13,0,'$\sigma_v=\pm 400 [\frac{\mu m}{s}]$','interpreter','latex','fontsize',14);
title('$\bf{x_0+v_0*t+\frac{1}{2}*a*t^2}$','interpreter','latex','fontsize',16)
```

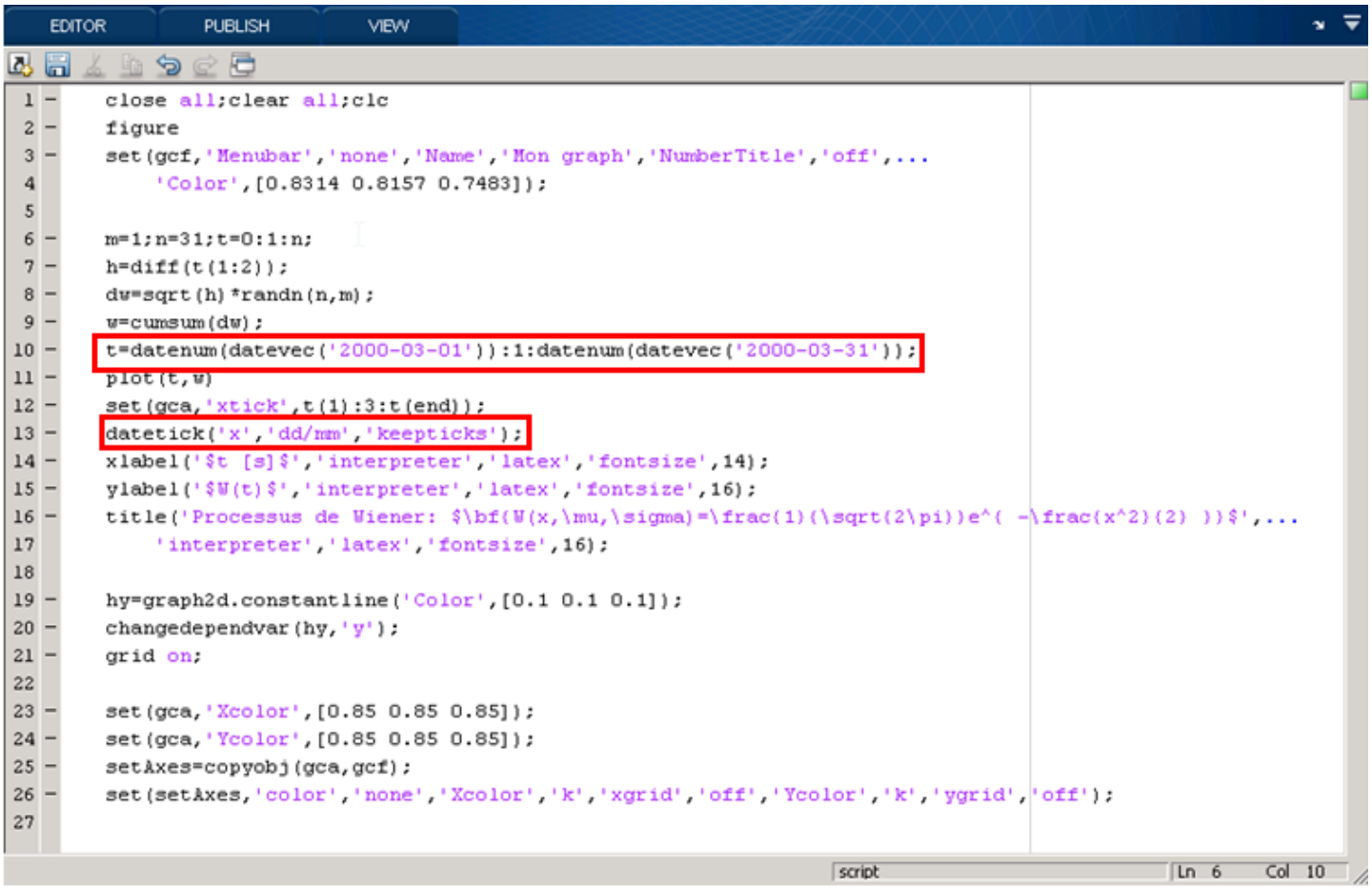
This will give the following result:



Plots with dates in axis

Managing dates is almost in all softwares an Art because of the lack of respect of international norms. Even if you or your customers respect international norms it is still a nightmare to manage in international finance. There are books (PDFs) of 40 pages explaining how to manager dates and times in international trading... then I let you imagine...

Here is an example with a date vector created by hand. As you can see we use it we use a function named **datetick( )** to choose the format of the dates:

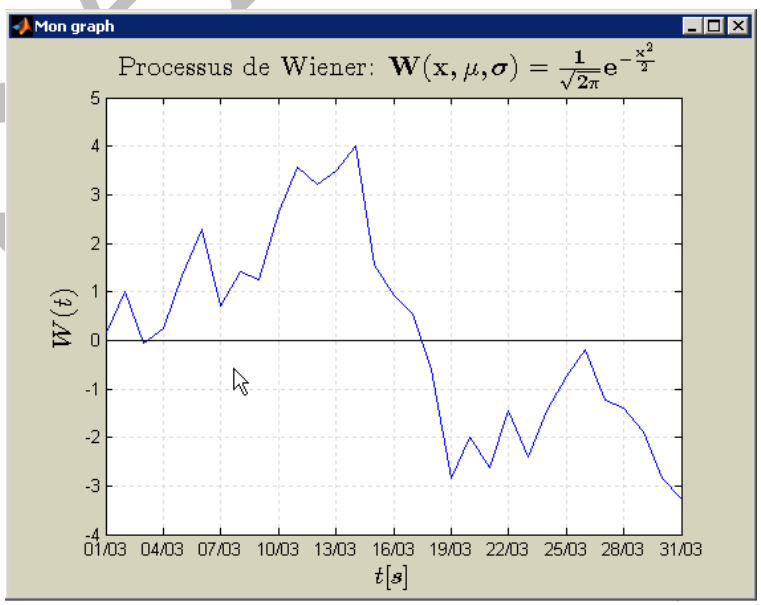


Here is a table with the most common used dates format:

Symbolic Identifier	Description	Example
yyyy	Year in full	1990, 2002
yy	Year in two digits	90, 02
QQ	Quarter year using letter Q and one digit	Q1
mmmm	Month using full name	March, December
mmm	Month using first three letters	Mar, Dec
mm	Month in two digits	03, 12
m	Month using capitalized first letter	M, D
dddd	Day using full name	Monday, Tuesday
ddd	Day using first three letters	Mon, Tue
dd	Day in two digits	05, 20
d	Day using capitalized first letter	M, T
HH	Hour in two digits (no leading zeros when symbolic identifier AM or PM is used)	05, 5 AM
MM	Minute in two digits	12, 02
SS	Second in two digits	07, 59
FFF	Millisecond in three digits	057
AM or PM	AM or PM inserted in date string	3:45:02 PM

Table 2 Common dates format syntaxes

The result will as following:

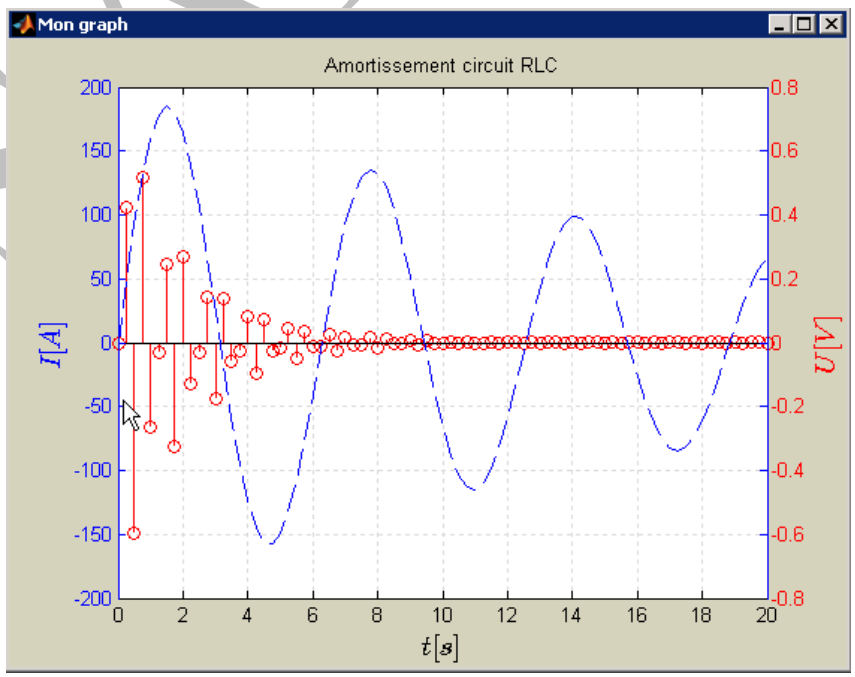


Multiple axis on line plots

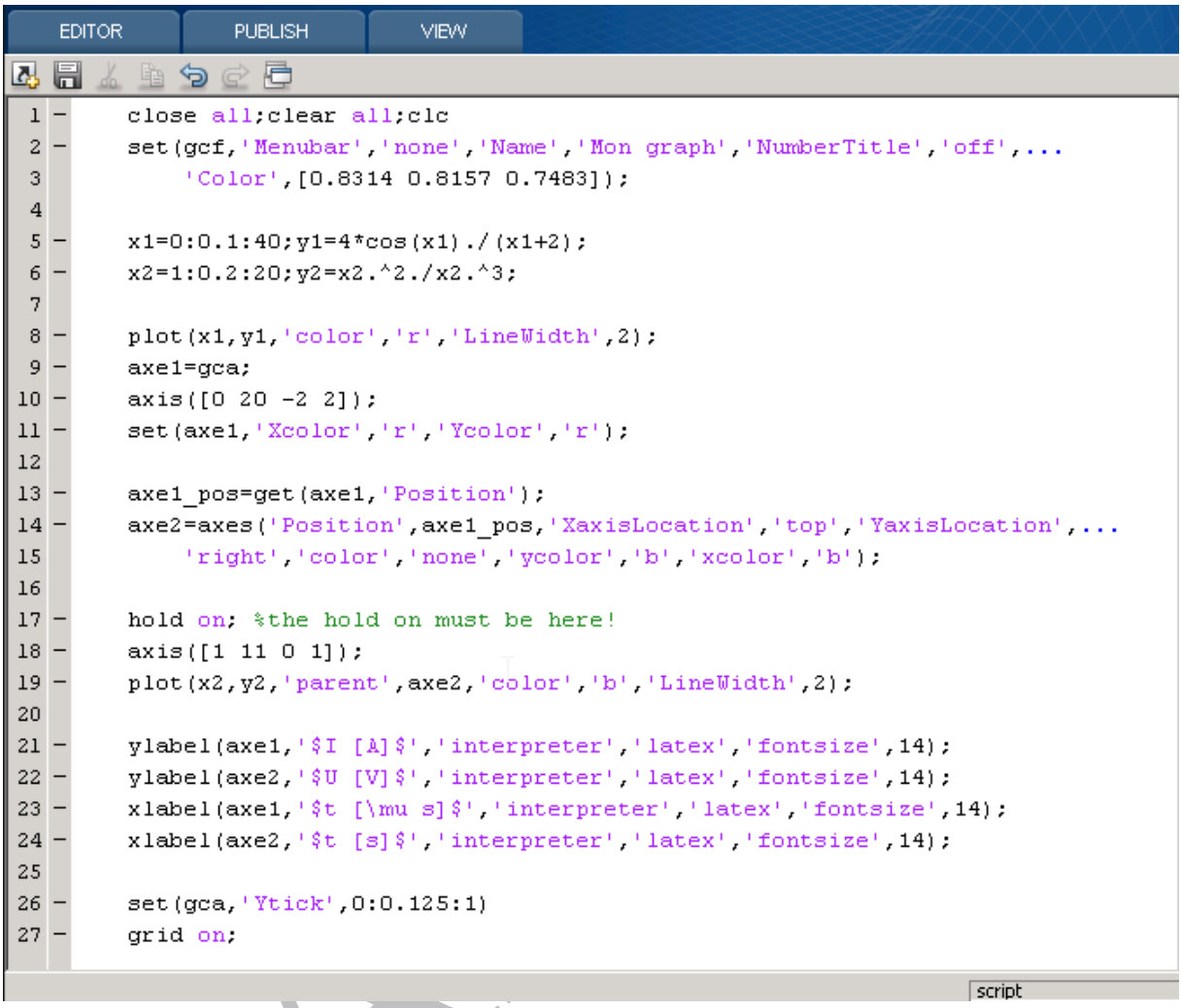
Multiple axes are a recurrent request in business usage or in physics and electronics. We will see here we can manage this in different ways. But first we create a simple double Y axes plot with the function **plotyy( )** and we change the color of the axes:

```
1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Color',[0.8314 0.8157 0.7483]);
5
6  t=0:0.25:20;
7  I=200*exp(-0.05*t).*sin(t);
8  U=0.8*exp(-0.5*t).*sin(10*t);
9  [ax, line1, line2]=plotyy(t,I,t,U,'plot','stem');
10
11 set(line1,'Color','b','LineStyle','--');
12 set(line2,'Color','r');
13 set(ax(2),'ycolor','r');
14
15 title('Amortissement circuit RLC');
16 xlabel('$t [s]$','interpreter','latex','fontsize',14);
17 ylabel(ax(1),'$I [A]$','interpreter','latex','fontsize',14);
18 ylabel(ax(2),'$U [V]$','interpreter','latex','fontsize',14);
19
20 grid on;
21
22 set(gca,'Xcolor',[0.85 0.85 0.85]);
23 set(gca,'Ycolor',[0.85 0.85 0.85]);
24 setAxes=copyobj(gca,gcf);
25 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','b','ygrid','off');
```

The results will looks as following:



If we want a rare 4 axes plot (until today only one customer in business asked me for that...) you have to use a lot of MATLAB tricks that are almost impossible to guess in my point of view and the adjustment of tick labels is always a pain in the ass:




```
close all;clear all;clc
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);

x1=0:0.1:40;y1=4*cos(x1)./(x1+2);
x2=1:0.2:20;y2=x2.^2./x2.^3;

plot(x1,y1,'color','r','LineWidth',2);
axe1=gca;
axis([0 20 -2 2]);
set(axe1,'Xcolor','r','Ycolor','r');

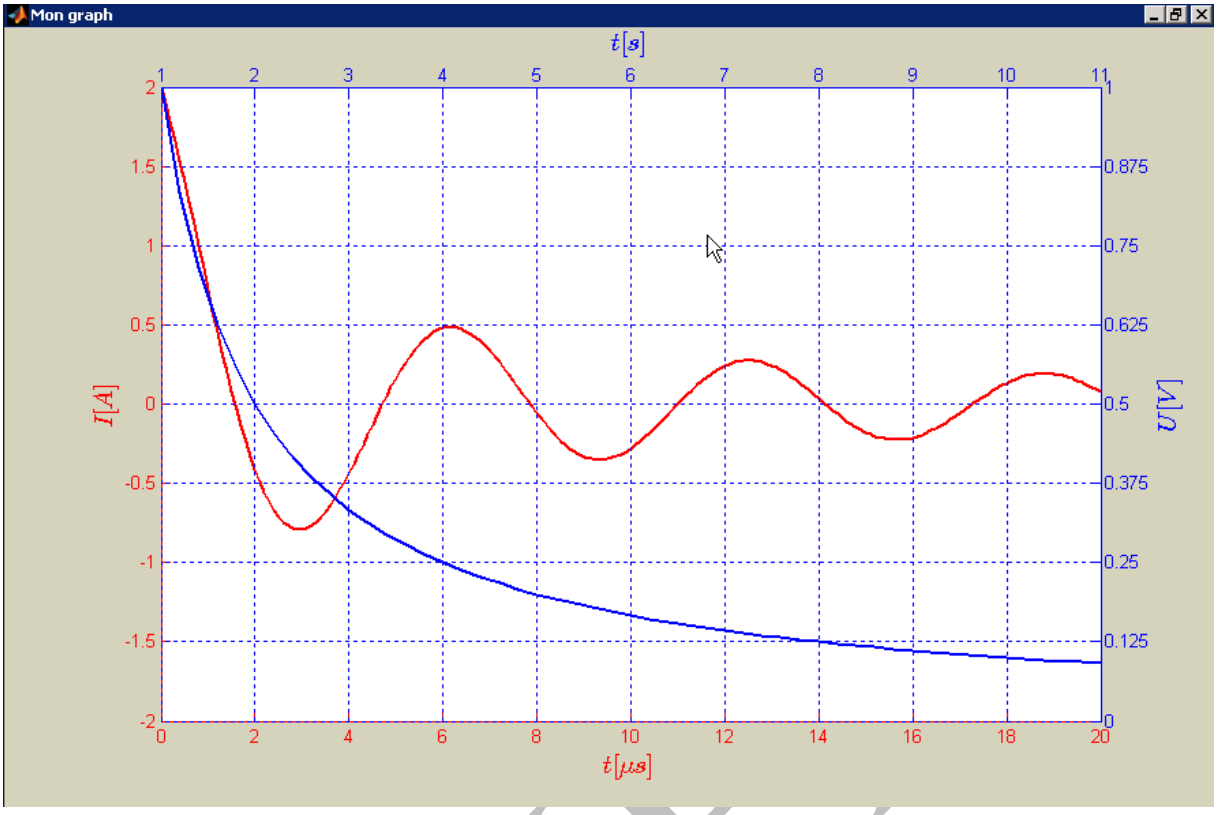
axe1_pos=get(axe1,'Position');
axe2=axes('Position',axe1_pos,'XaxisLocation','top','YaxisLocation',...
    'right','color','none','ycolor','b','xcolor','b');

hold on; %the hold on must be here!
axis([1 11 0 1]);
plot(x2,y2,'parent',axe2,'color','b','LineWidth',2);

ylabel(axe1,'$I [A]$','interpreter','latex','fontsize',14);
ylabel(axe2,'$U [V]$','interpreter','latex','fontsize',14);
xlabel(axe1,'$t [\mu s]$','interpreter','latex','fontsize',14);
xlabel(axe2,'$t [s]$','interpreter','latex','fontsize',14);

set(gca,'Ytick',0:0.125:1)
grid on;
```

SA



SAMPLE

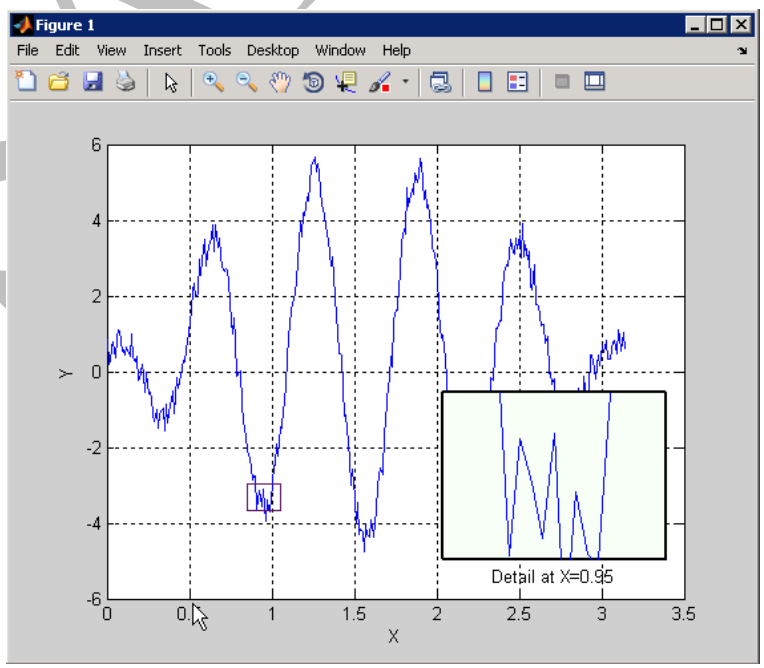
Magnifier on line plot

Create a magnifier is a wonderful recapitulative exercise for almost all what we have seen until now about charts in MATLAB. Let see how to do this and the result:

```
% Create a dummy function
x = 0:0.01:pi;
y = 5*sin(x).*cos(10*x) + rand(size(x));
plot(x, y); grid on;
xlabel('X');
ylabel('Y');

% Specify the position and the size of the rectangle
x_r = 0.95; y_r = -3.3; w_r = 0.2; h_r = 0.7;
rectangle('Position', [x_r-w_r/2, y_r-h_r/2, w_r, h_r], ...
          'EdgeColor', [0.4, 0.1, 0.4]);

% We specify the position and the size of the second axis
% The position is in percentage of the whole plot area
x_a = 0.58; y_a = 0.18; w_a = 0.3; h_a = 0.3;
ax = axes('Units', 'Normalized', ...
          'Position', [x_a, y_a, w_a, h_a], ...
          'XTick', [], ... %no ticks
          'YTick', [], ... %no ticks
          'Box', 'on', ...
          'LineWidth', 2, ...
          'Color', [0.95, 0.99, 0.95]);
hold on;
plot(x, y);
lbl=xlabel(strcat('Detail at X=',num2str(x_r)));
axis([x_r-w_r/2, x_r+w_r/2, y_r-h_r/2, y_r+h_r/2]);
```

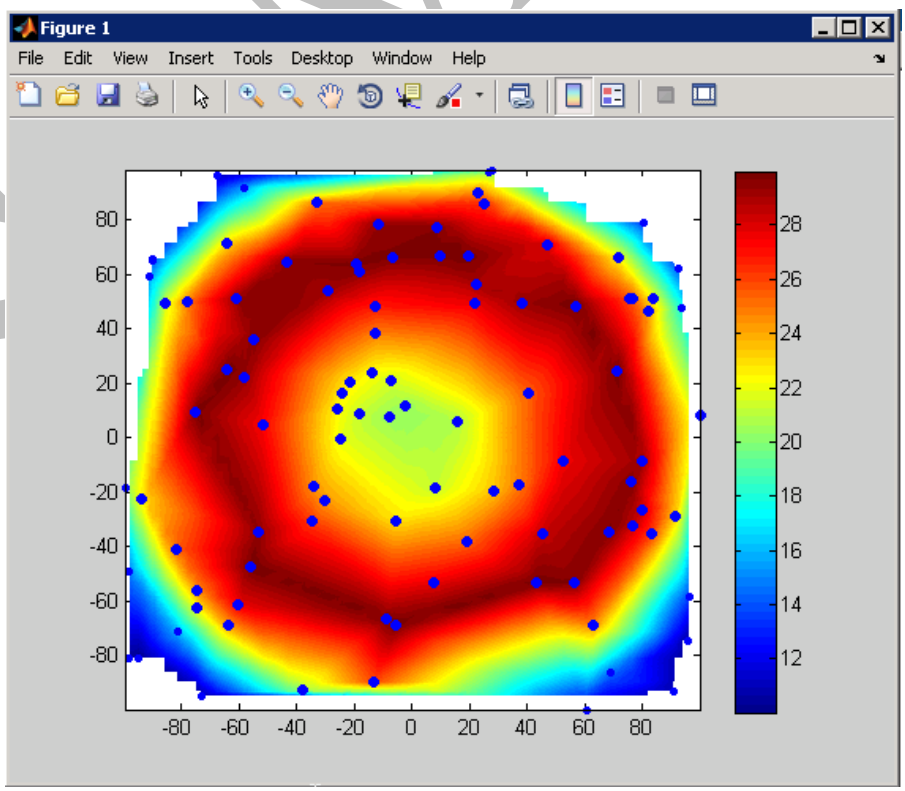


Points density

The following chart is not only used in association with a map in background to show the density of animals, humans or any other stuff in an area or clicks on a web page perimeter but also in data mining (see also the available plots for another method using the statistical toolbox on page 772 and 773)

```
%Lon coordinates
x=rand(100,1)*200-100;
 %Lat coordinates
y=rand(100,1)*200-100;
%Faux-measured data
z=10*sin((x*pi/180).^2+(y*pi/180).^2)+20;
%Three steps of magic
[X,Y]=meshgrid(unique(x),unique(y));
Z=griddata(x,y,z,X,Y);
pcolor(X,Y,Z);
shading interp;
hold on;
%Bubble plot of 3D data
scatter(x,y,z,'filled');
colorbar;
```

This gives:



2D vertical/horizontal bar charts

2D vertical bar charts are used a lot in business, statistics and finance. We will see here the most common usage of this type of charts and once again give a lot of tips to customize them because it is not as easy as in a spreadsheet software like Microsoft Excel.

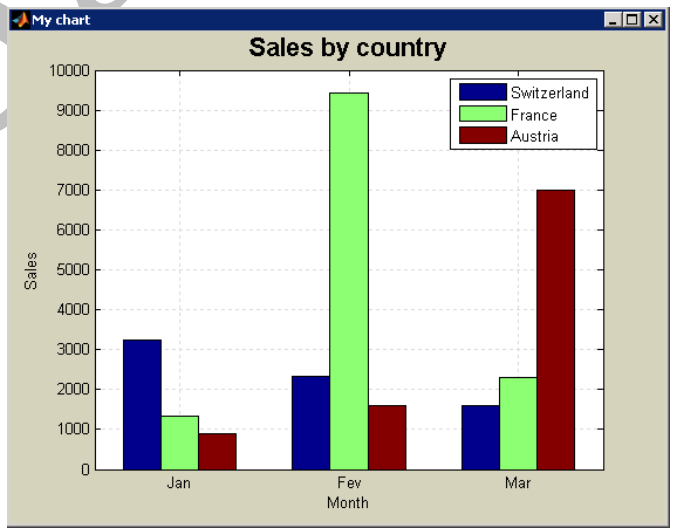
Multiple series bar chart

First we begin with a very common task... Plot multiple series of data on a charts using the function **bar( )** whose last parameter is the distance between the groups:

```
1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','My chart','NumberTitle','off',...
4      'Color',[0.8314 0.8157, 0.7483]);
5
6  Switzerland=[3233 2323 1600];
7  France=[1340 9432 2304];
8  Austria=[900 1594 7000];
9  bar(1:3,[Switzerland' France' Austria'],1)
10 set(gca,'Xtick',1:3,'xTickLabel',{'Jan', 'Fev','Mar'});
11 legend('Switzerland','France','Austria');
12 xlabel('Month');ylabel('Sales');
13 title('Sales by country','fontweight','bold','fontsize',16);
14
15 grid on
16
17 set(gca,'Xcolor',[0.85 0.85 0.85]);
18 set(gca,'Ycolor',[0.85 0.85 0.85]);
19 setAxes=copyobj(gca,gcf);
20 set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
```

You can see at the end the trick to keep the major axis visible after we have customized the grid (this is a small difficulty with bar charts...).

The result looks like this:



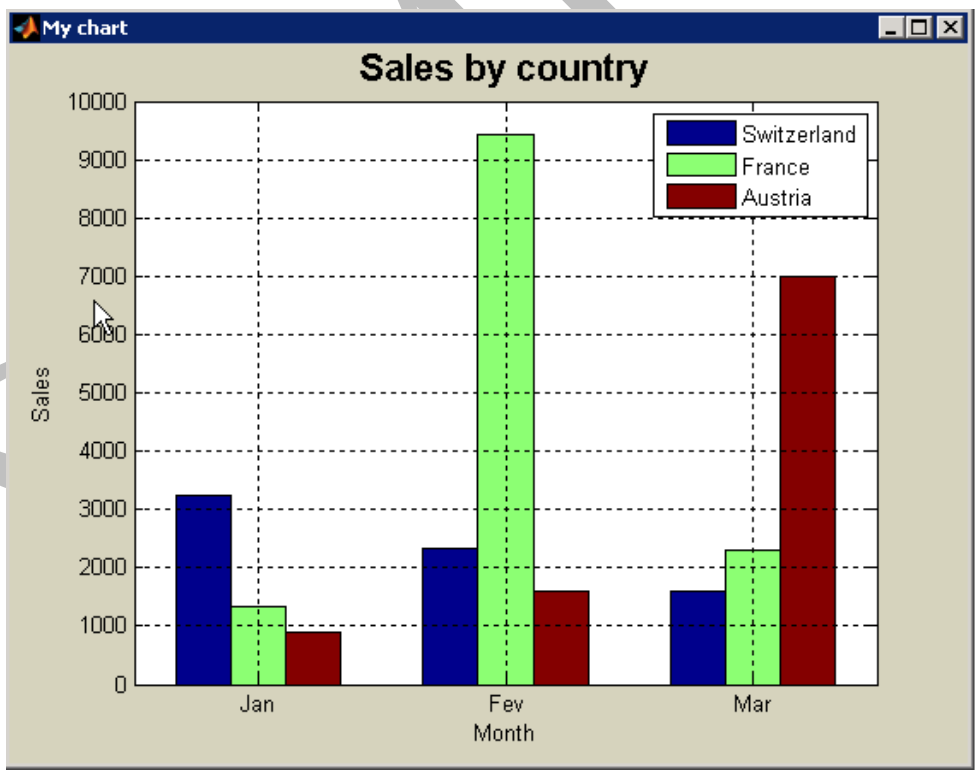
If you want the gridline on the top you just need to write the following:

```
close all;clear all;clc
figure
set(gcf,'Menubar','none','Name','My chart','NumberTitle','off',...
    'Color',[0.8314 0.8157, 0.7483]);

Switzerland=[3233 2323 1600];
France=[1340 9432 2304];
Austria=[900 1594 7000];
bar(1:3,[Switzerland' France' Austria'],1)
set(gca,'Xtick',1:3,'xTickLabel',{'Jan', 'Fev','Mar'});
legend('Switzerland','France','Austria');
xlabel('Month');ylabel('Sales');
title('Sales by country','fontweight','bold','fontsize',16);

grid on
set(gca,'Layer','top');
```

You will get:

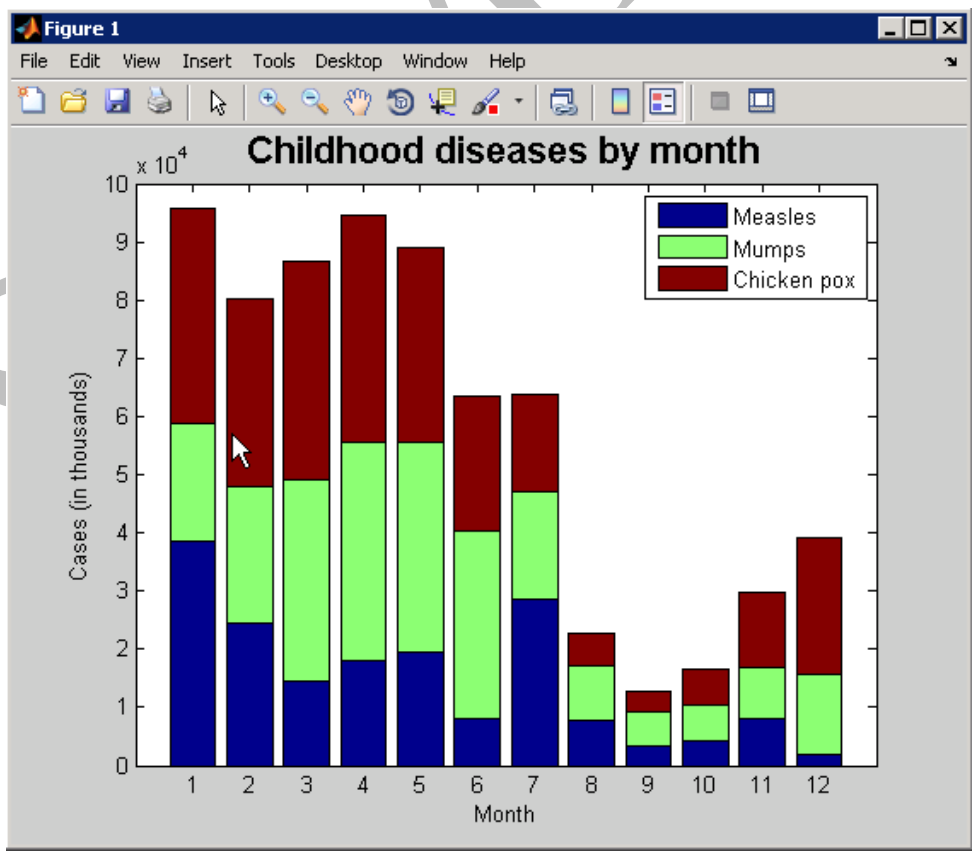


Stacked bar chart

Another typical common chart in spreadsheet softwares is to create a stacked bar chart. This is very easy we just have to change the last parameter of the function bar( ) as you can see below:

```
%plot 2d histogramme groupé
%Pour normaliser il faut diviser (il n'existe pas de bar chart normalisé
%dans Matlab)
measles = [38556 24472 14556 18060 19549 8122 28541 7880 3283 4135 7953 1884];
mumps = [20178 23536 34561 37395 36072 32237 18597 9408 6005 6268 8963 13882];
chickenPox = [37140 32169 37533 39103 33244 23269 16737 5411 3435 6052 12825 23332];
figure;
bar(1:12, [measles' mumps' chickenPox'], 'stack');
axis([0 13 0 100000]);
title('Childhood diseases by month','fontweight','bold','fontsize',16);
xlabel('Month');
ylabel('Cases (in thousands)');
legend('Measles', 'Mumps', 'Chicken pox');
[im_hatch,colorlist] = applyhatch_pluscolor(1,'|-.x',1,[1 0 1 1]);
```

The result will be:



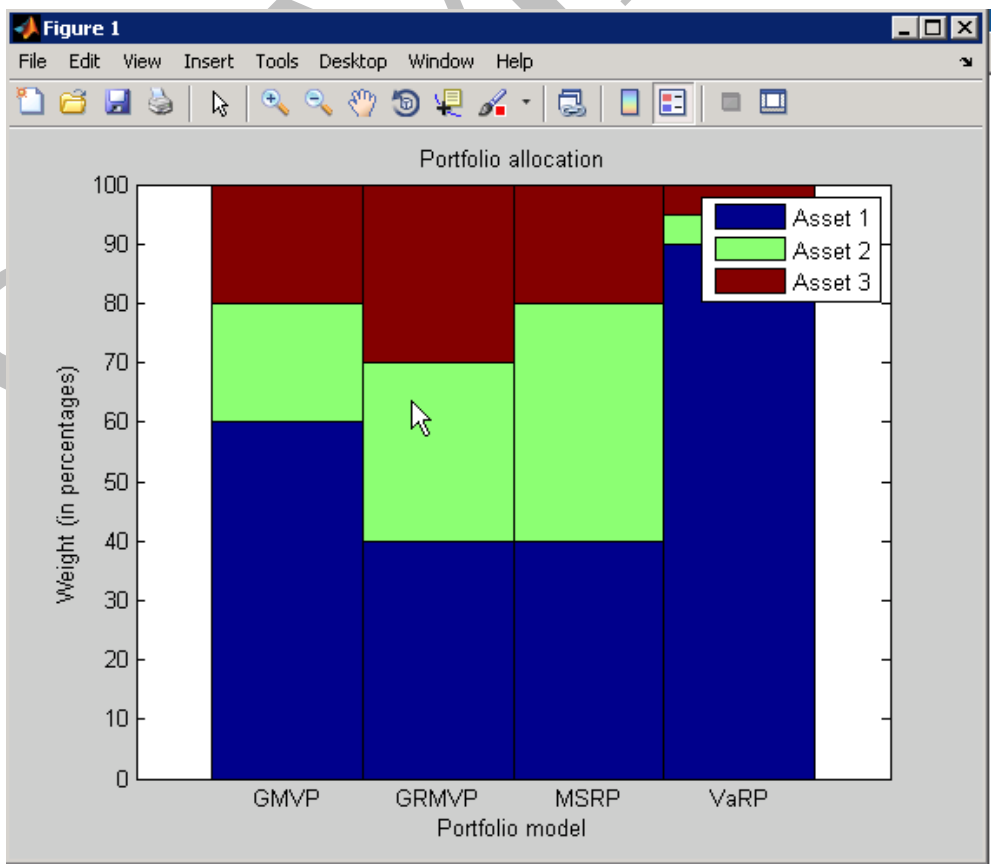
Stacked normalized bar chart

Even more frequent as stacked bar chart and use a lot in finance especially in the field of portfolio allocation, we have the stacked normalized bar chart.

Because this type of chart doesn't exists in MATLAB you will always have to normalized you vectors yourself (simple arithmetic). The only trick to know otherwise is to give the parameter "1" to the **bar( )** function to have a null distance between column bars:

```
1   asset_1 = [0.6 0.4 0.4 0.9]';
2   asset_2 = [0.2 0.3 0.4 0.05]';
3   asset_3 = [0.2 0.3 0.2 0.05]';
4   figure;
5   bar(1:4, [asset_1*100 asset_2*100 asset_3*100], 1, 'stack');
6   axis([0 5 0 100]);
7   title('Portfolio allocation');
8   xlabel('Portfolio model');
9   ylabel('Weight (in percentages)');
10  legend('Asset 1', 'Asset 2', 'Asset 3');
11  set(gca,'XTickLabel',{'GMVP', 'GRMVP', 'MSRP', 'VaRP'})
```

The result will look as following:

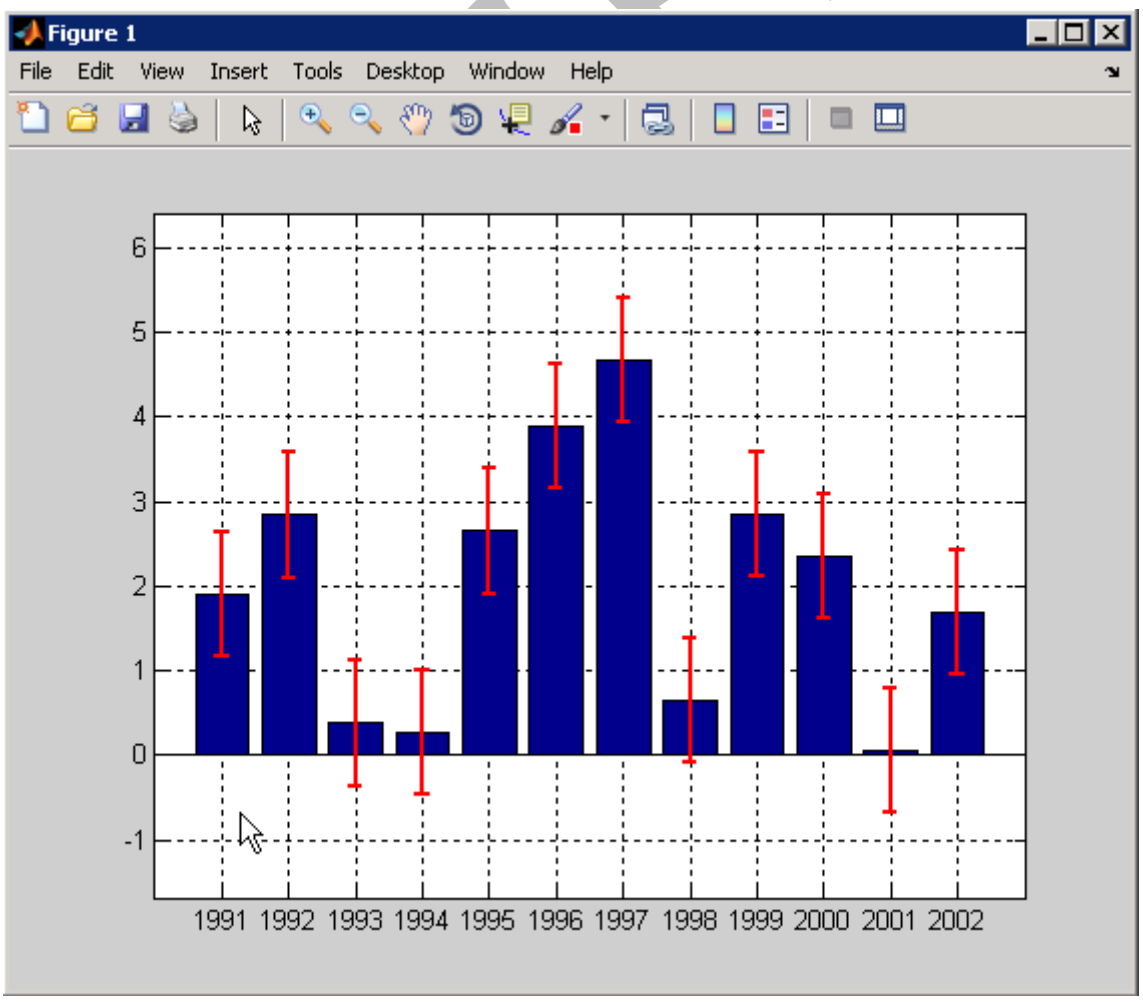


Bar chart with error bars

Now a very common bar chart in physics and statistics is this with error bars at the top of the bars. To do this it is simple if we mix our knowledge of the function **bar( )** and **errorbar( )** but we have to add a line at the end to manage an awful behavior of MATLAB with *X* and *Y* in such a situation:

```
1  -    clear all;clc;close all
2  -    intYears=12;
3  -    y=5*rand(intYears,1);
4  -    e=std(y/2)*ones(size(y));
5  -    x=1991:1:1991+intYears-1;
6  -    bar(x,y);
7  -    hold on;
8  -    grid on;
9  -    h=errorbar(x,y,e);
10 -    set(h(1),'color','r','LineStyle','none','LineWidth',2);
11      %we are obliged to manage axes range otherwise the chart is sometimes awful
12 -    ylim([min(y-e)-1 max(y+e)+1]);xlim([(1991)-1 (1991+intYears-1)+1])
```

The result looks like:

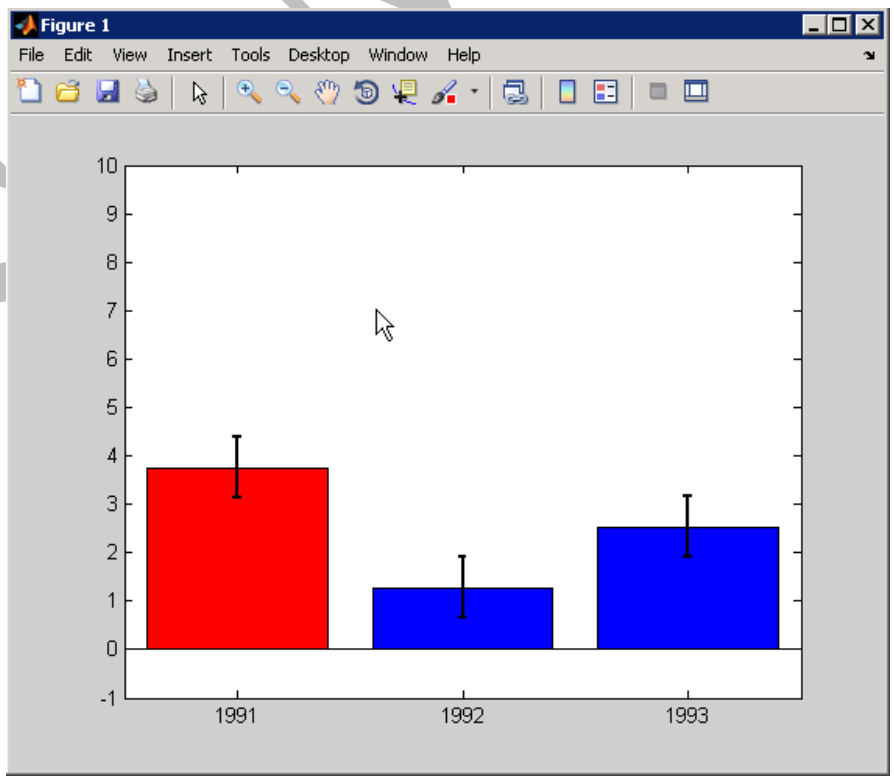


Bar chart with bar having a special a color

To change the color of only one bar the manipulation also difficult to guess. We have to build the bar chart bar by bar... as you can see here below:

```
clear all;clc;close all
y=5*rand(3,1);
e=std(y/2)*ones(size(y));
x=1991:1993;
for i=1:length(y)
    if y(i)>3.5
        colorcode = 'r';
    else
        colorcode = 'b';
    end
    bar(i, y(i), colorcode);
    hold on;
end
set(gca,'Xtick',1:3,'XTickLabel',x)
h=errorbar(1:3,y,e);
set(h(1),'color','k','LineStyle','none','LineWidth',2);
ylim([-1 10]);
```

The result will looks as following:



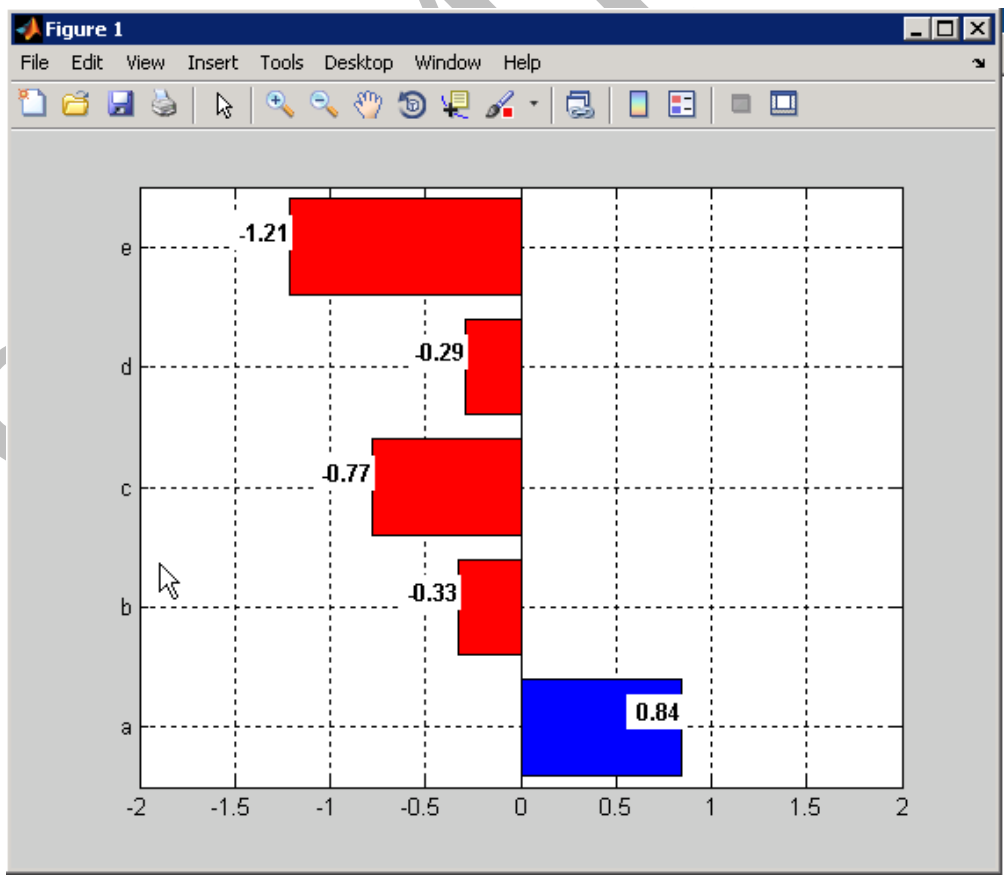
Tornado Chart

Tornados chart are used a lot in finance this is why we have to see how to create such a chart.

First we will have to use the **barh( )** function after to use a colormap trick for negative/positive values et set ticks manually and also to position values "manually" on the top of the bars as you can see bellow:

```
1  -    y=rand(5,1)*3-1.5;
2
3  -    hb=barh(y);
4  -    set(get(hb,'children'),'cdata', sign(y) );
5  -    colormap([1 0 0; 0 0 1]); % red & blue in rgb
6  -    set(gca,'Ytick',1:5,'yTickLabel',{'a'; 'b'; 'c'; 'd';'e'})
7  -    text(y,1:5,num2str(y,'%0.2f'),'HorizontalAlignment','right',...
8           'VerticalAlignment','bottom','FontWeight','bold',...
9           'BackgroundColor',[1 1 1])
10   -    xlim([-2 2]);
11   -    grid on;
```

The result will be:



Bar chart with line and multiple axis

Sometimes users want to mix a chart line with a line chart for readiness purposes. For this purpose you can use the function **plotyy( )** that not only management multiple axes but also multiple chart type as you can see below:

```
TBdata = [1990 4889 16.4; 1991 5273 17.4; 1992 5382 17.4; 1993 5173 16.5;
          1994 4860 15.4; 1995 4675 14.7; 1996 4313 13.5; 1997 4059 12.5;
          1998 3855 11.7; 1999 3608 10.8; 2000 3297  9.7; 2001 3332  9.6;
          2002 3169  9.0; 2003 3227  9.0; 2004 2989  8.2; 2005 2903  7.9;
          2006 2779  7.4; 2007 2725  7.2];

years = TBdata(:,1);
cases = TBdata(:,2);
rate  = TBdata(:,3);

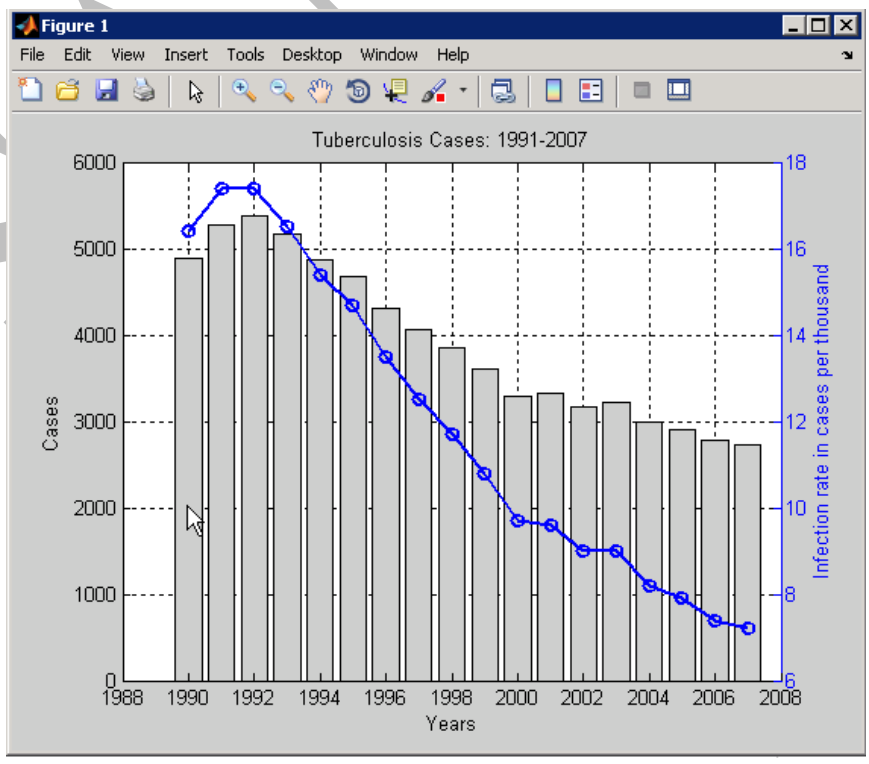
[ax, h1, h2] = plotyy(years, cases, years, rate, 'bar', 'plot');

set(h1, 'FaceColor', [0.8, 0.8, 0.8]);
set(h2, 'LineWidth', 2,'Marker','o');
grid on;

title('Tuberculosis Cases: 1991-2007');
xlabel('Years');

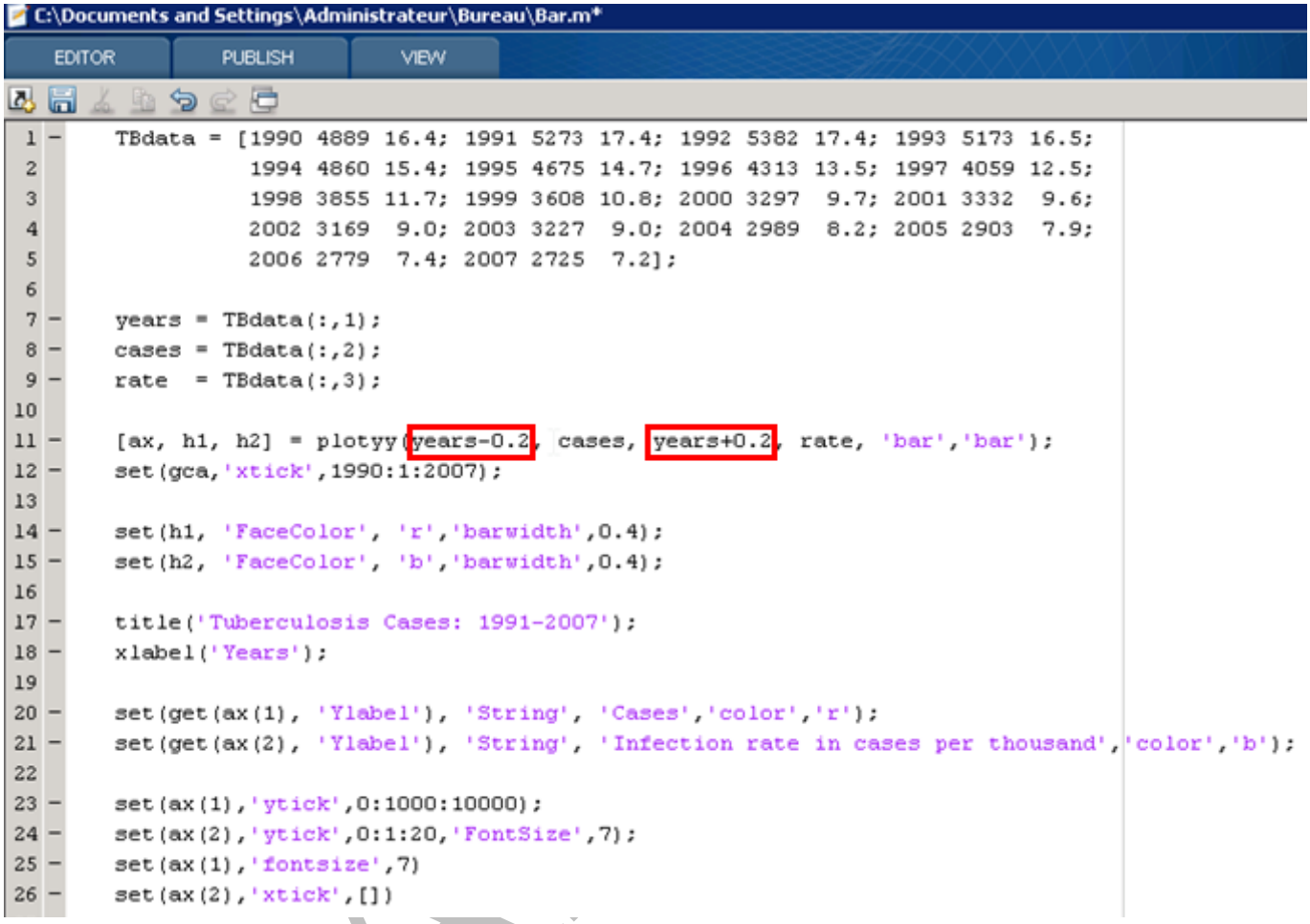
set(get(ax(1), 'Ylabel'), 'String', 'Cases');
set(get(ax(2), 'Ylabel'), 'String', 'Infection rate in cases per thousand');
```

The result will be:



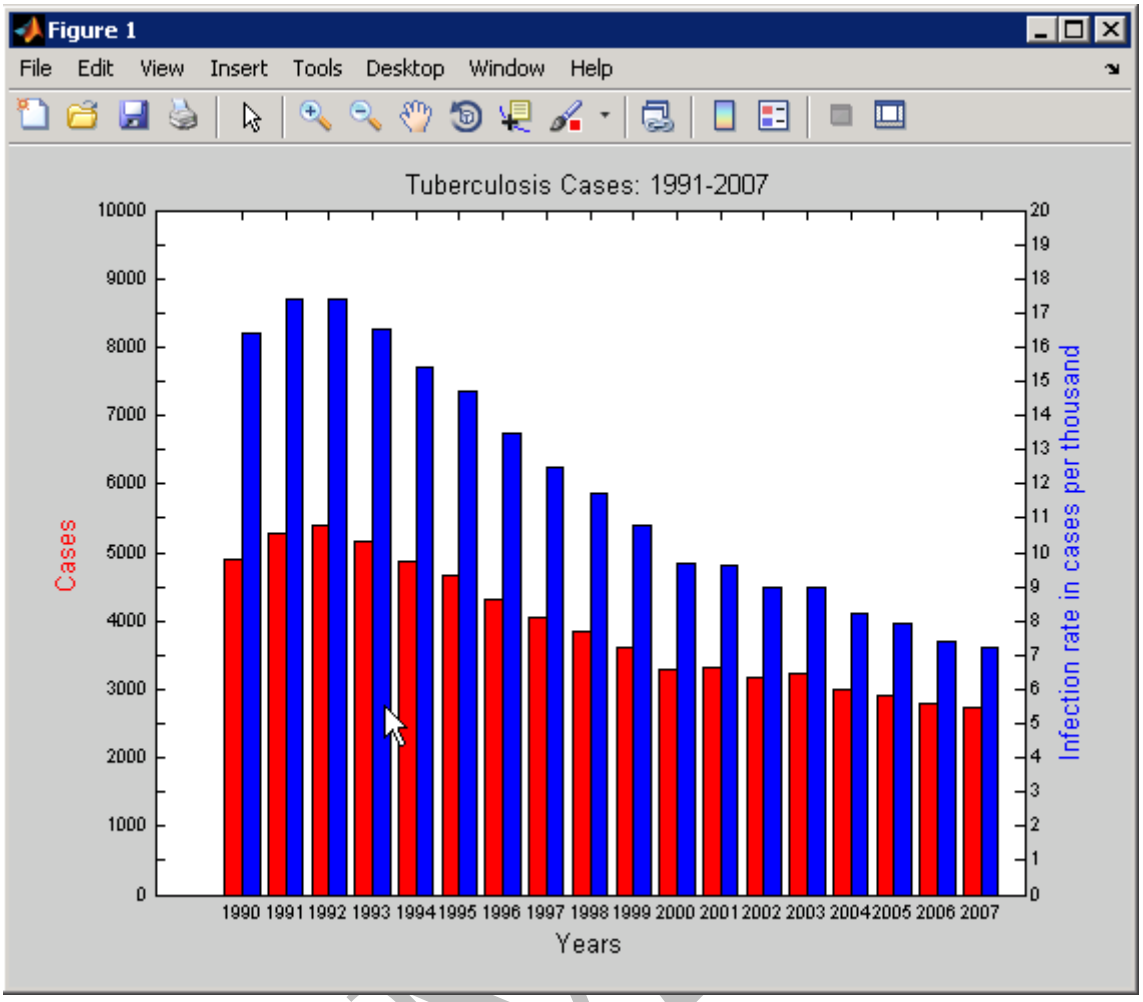
Two series bar chart with multiple axis

Another variant of the previous example is a multiple axis plot with two bar charts. For sure we will still use the **plotyy( )** but MATLAB has a problem managing the same abscises values. Then we have to use a trick:




```
1  -    TBdata = [1990 4889 16.4; 1991 5273 17.4; 1992 5382 17.4; 1993 5173 16.5;
2               1994 4860 15.4; 1995 4675 14.7; 1996 4313 13.5; 1997 4059 12.5;
3               1998 3855 11.7; 1999 3608 10.8; 2000 3297  9.7; 2001 3332  9.6;
4               2002 3169  9.0; 2003 3227  9.0; 2004 2989  8.2; 2005 2903  7.9;
5               2006 2779  7.4; 2007 2725  7.2];
6
7  -    years = TBdata(:,1);
8  -    cases = TBdata(:,2);
9  -    rate  = TBdata(:,3);
10
11 -    [ax, h1, h2] = plotyy(years-0.2, cases, years+0.2, rate, 'bar','bar');
12 -    set(gca,'xtick',1990:1:2007);
13
14 -    set(h1, 'FaceColor', 'r','barwidth',0.4);
15 -    set(h2, 'FaceColor', 'b','barwidth',0.4);
16
17 -    title('Tuberculosis Cases: 1991-2007');
18 -    xlabel('Years');
19
20 -    set(get(ax(1), 'Ylabel'), 'String', 'Cases','color','r');
21 -    set(get(ax(2), 'Ylabel'), 'String', 'Infection rate in cases per thousand','color','b');
22
23 -    set(ax(1),'ytick',0:1000:10000);
24 -    set(ax(2),'ytick',0:1:20,'FontSize',7);
25 -    set(ax(1),'fontsize',7)
26 -    set(ax(2),'xtick',[])
```

The result is on next page:


Figure 1
File Edit View Insert Tools Desktop Window Help
Tuberculosis Cases: 1991-2007
Cases
Infection rate in cases per thousand
Years
0 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007

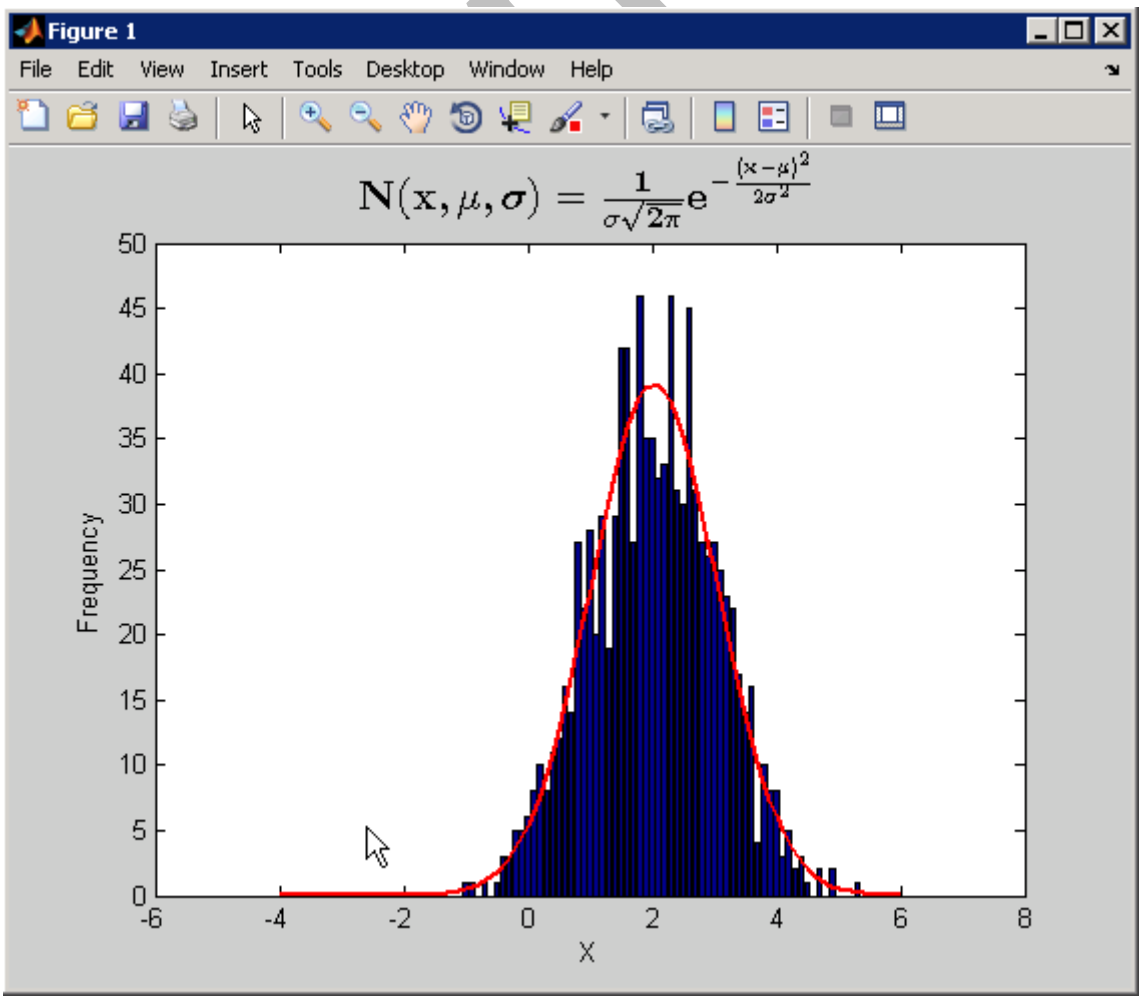
SAM

Probability histogram with density curve (pdf)

This is one on the most used bar chart type in Statistics. To make a PDF (probability density function) of a random variable we need to use the two functions **hist( )** and **bar( )** together:

```
%variable aléatoire normale centrée d'espérance 2 et d'écart-type unitaire
y=randn(1000,1) + 2; %le +2 est pour décaler la moyenne à la valeur de 2
x=-4:0.1:6;
n=hist(y,x);
bar(x,n);
MEAN=mean(y);
STD=sqrt(mean((y-MEAN).^2));
f=(1/(STD*sqrt(2*pi)))*exp(-0.5*((x-MEAN)/STD).^2);
f=f*sum(n)/sum(f);
hold on
plot(x,f,'r','LineWidth',2);
xlabel('X');ylabel('Frequency');
title('$\bf{N(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}$',...
    'interpreter','latex','fontsize',16);
```

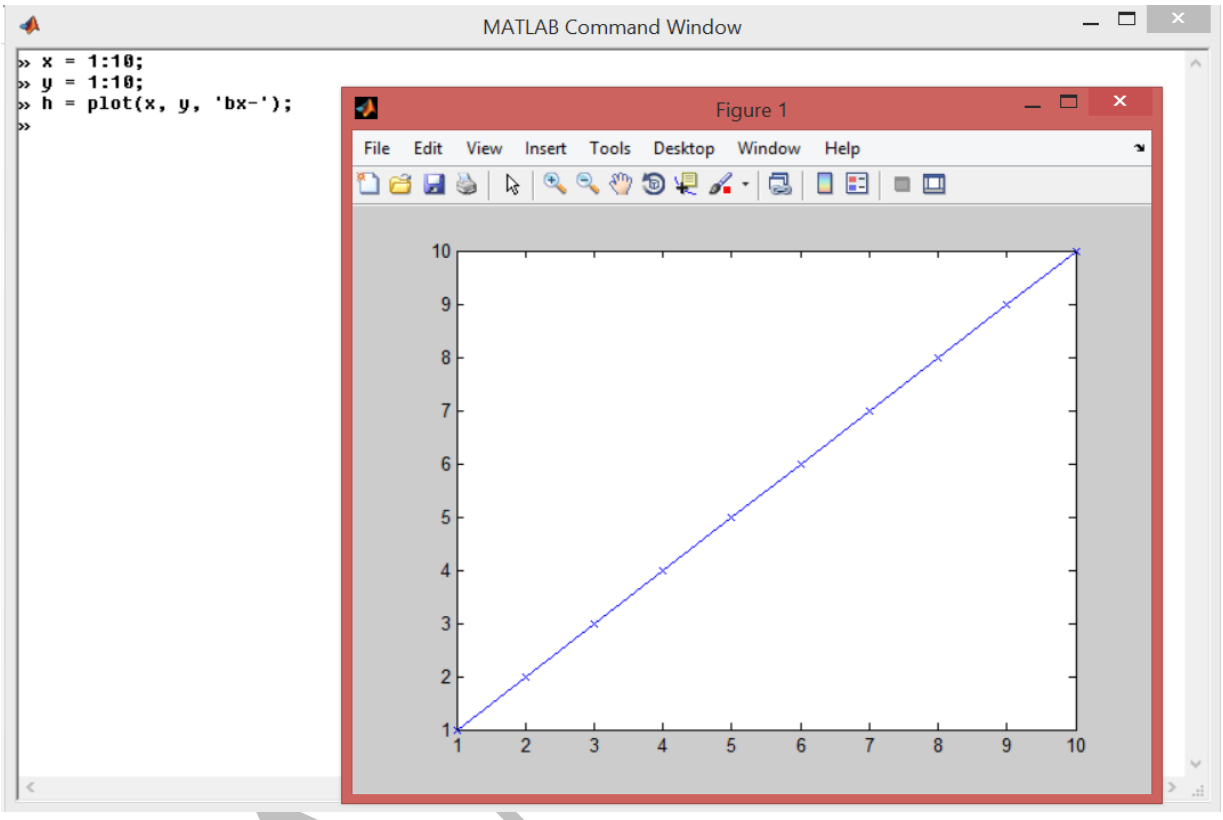
The result will be:



Handle charts

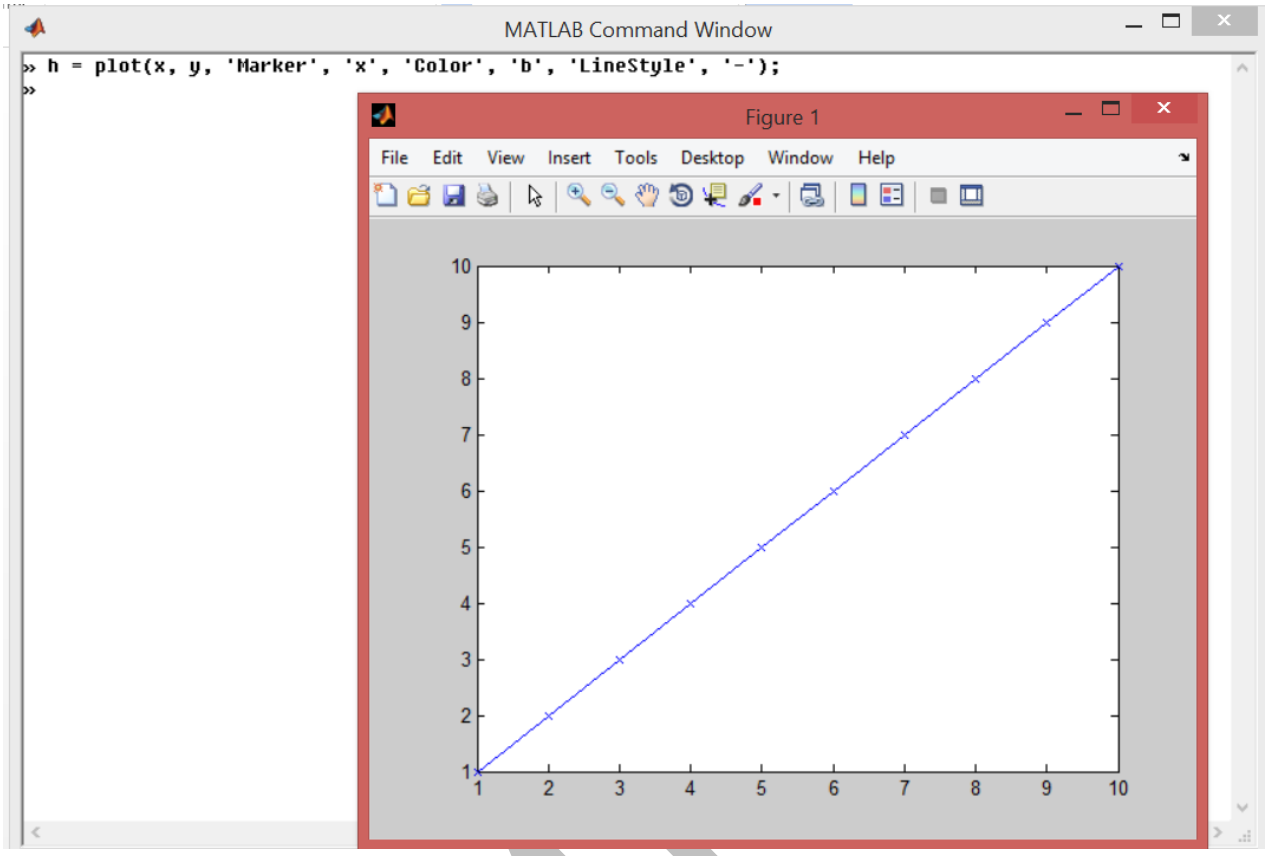
Handle graphics provide you with a rich set of functions and properties for generating imagery using MATLAB. Here's a brief resume to handle graphics for creating publication quality plots.

Consider the following example:



This will plot a series of 10 blue *x*'s, connected by a solid blue line and assigns a handle, in the form of a number, to the object. This handle can then be used to view or modify the object's properties. Storing the handle in a variable (h, in this case) allows for easier access to the object.

We can create the same plot more explicitly by specifying the properties we want the object to have after creation:



This also plots 10 blue *x*'s, connected by a solid blue line.

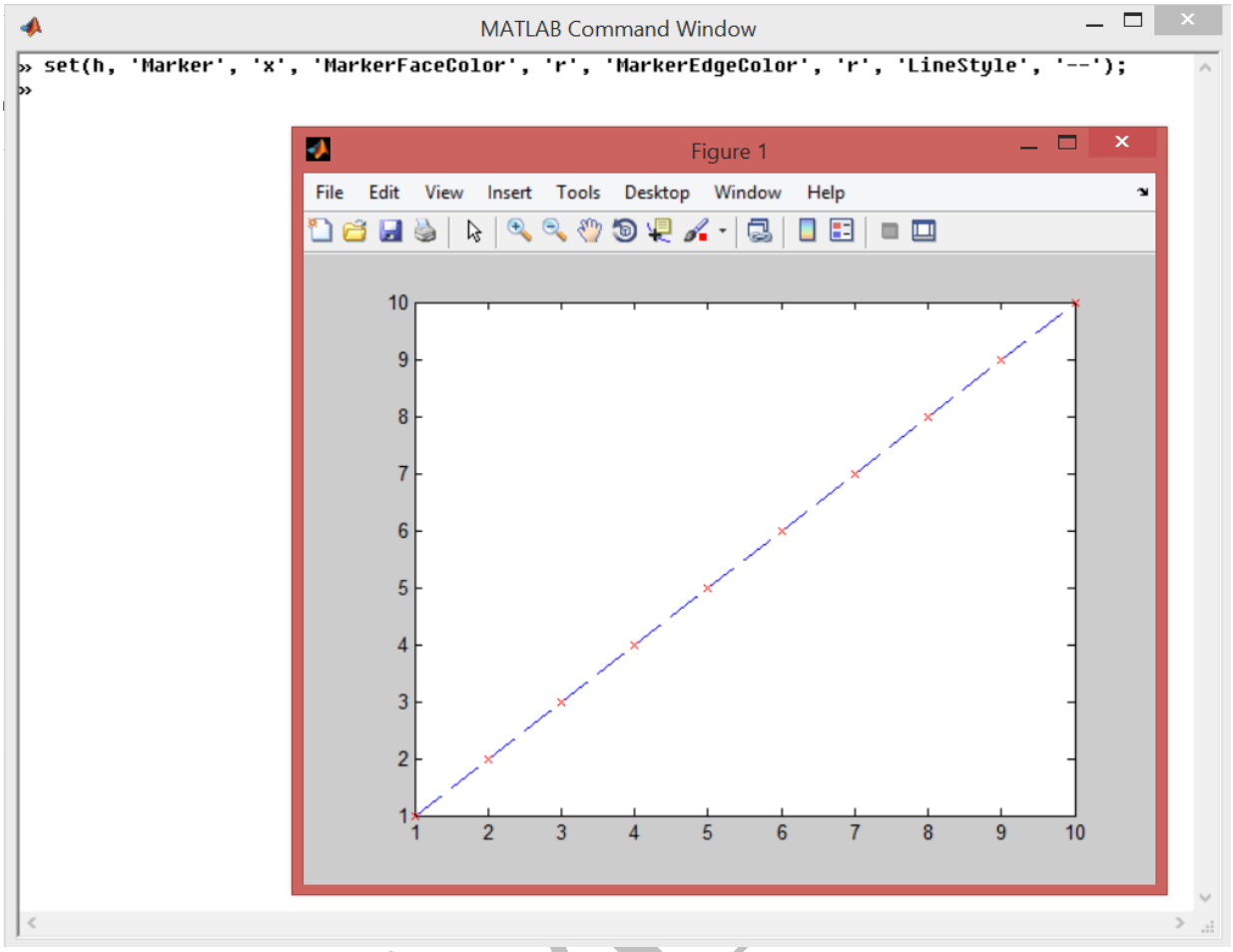
What if we want to view all of the properties of the line object after we plot it? We can use the MATLAB method **get( )** to view them:

SAMPLE



This displays all of the properties of the line we just plotted. Most of the properties can be changed using set, while a few are read-only.

If we want to change the blue *x*'s to filled red circles connected by a dashed line, we would do the following:

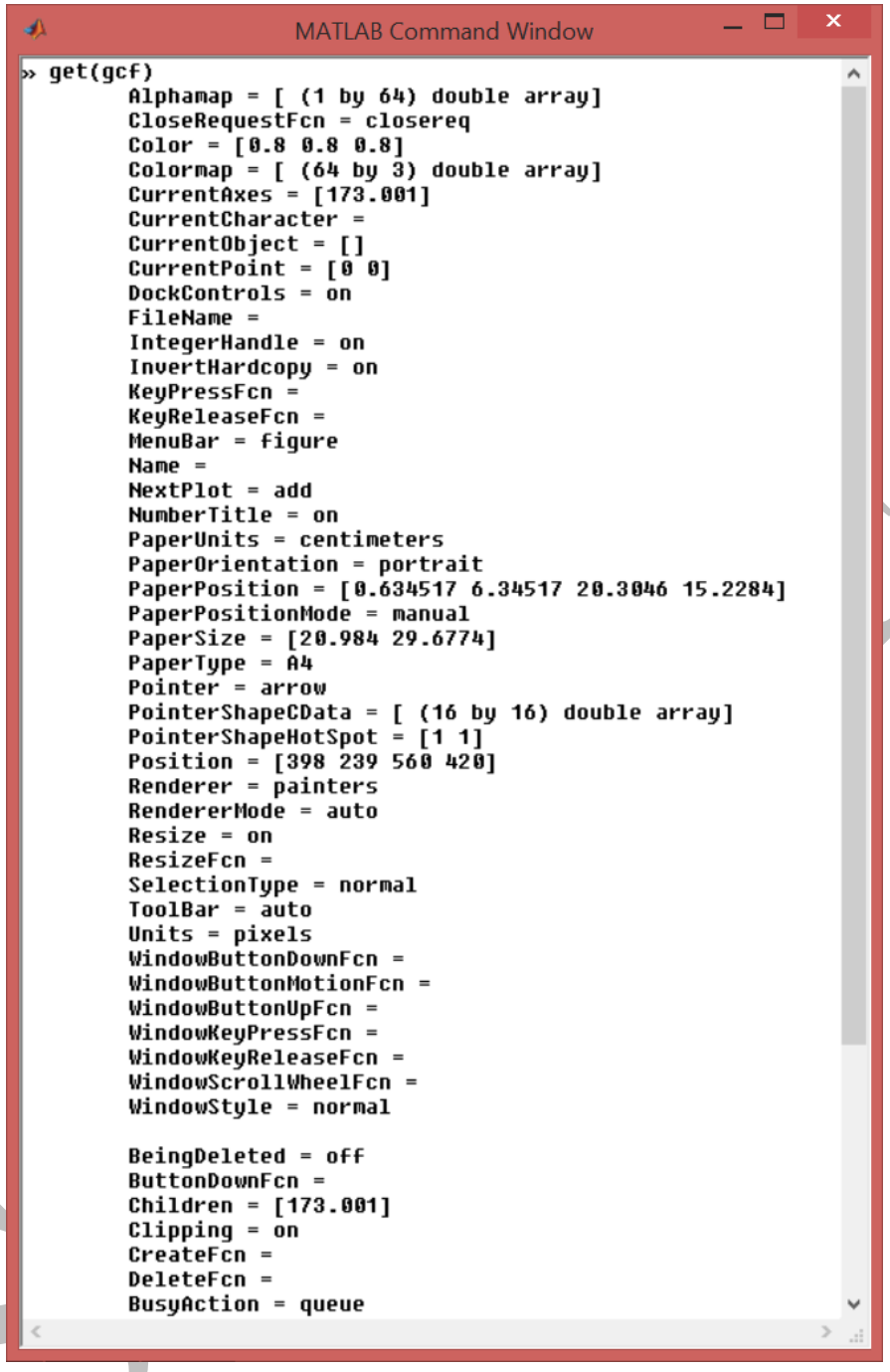


While assigning a graphic's object to a variable simplifies modifying the object after creation, there are ways to access a graphics object even if you forget. MATLAB provides access to recently plotted or accessed objects through a number the pre-defined variables *gcf* (handle to the current figure), *gca* (handle to the current axis) and *gco* (handle to the current object, which is almost always the graphics object created as the result of the last graphics command).

Here's how they work:

Let's say we want to look at the properties of the newly created figure window. I can use get, but I need to specify the handle of the figure window, which we did not explicitly store in a variable. MATLAB's pre-defined *gcf* (get current figure) contains the handle to the most recently accessed figure:

SAMPLE

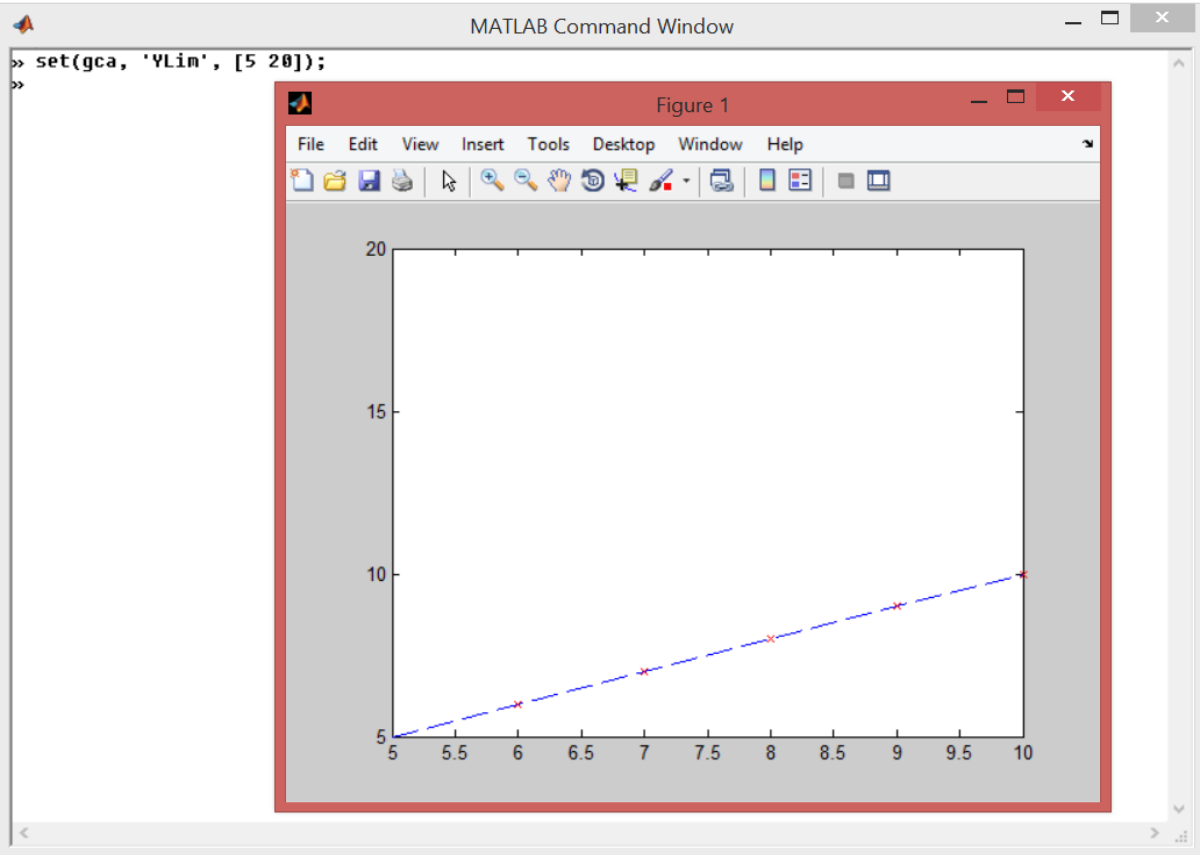


As we already know we can change the figure window properties using also the **set( )** method.

The same thing goes for the axes we just plotted the line on. We didn't explicitly store the handle to the axes in a variable, so we can use *gca* (get current axis) to access it:



As we know we can modify the bounds of the *y*-axis:

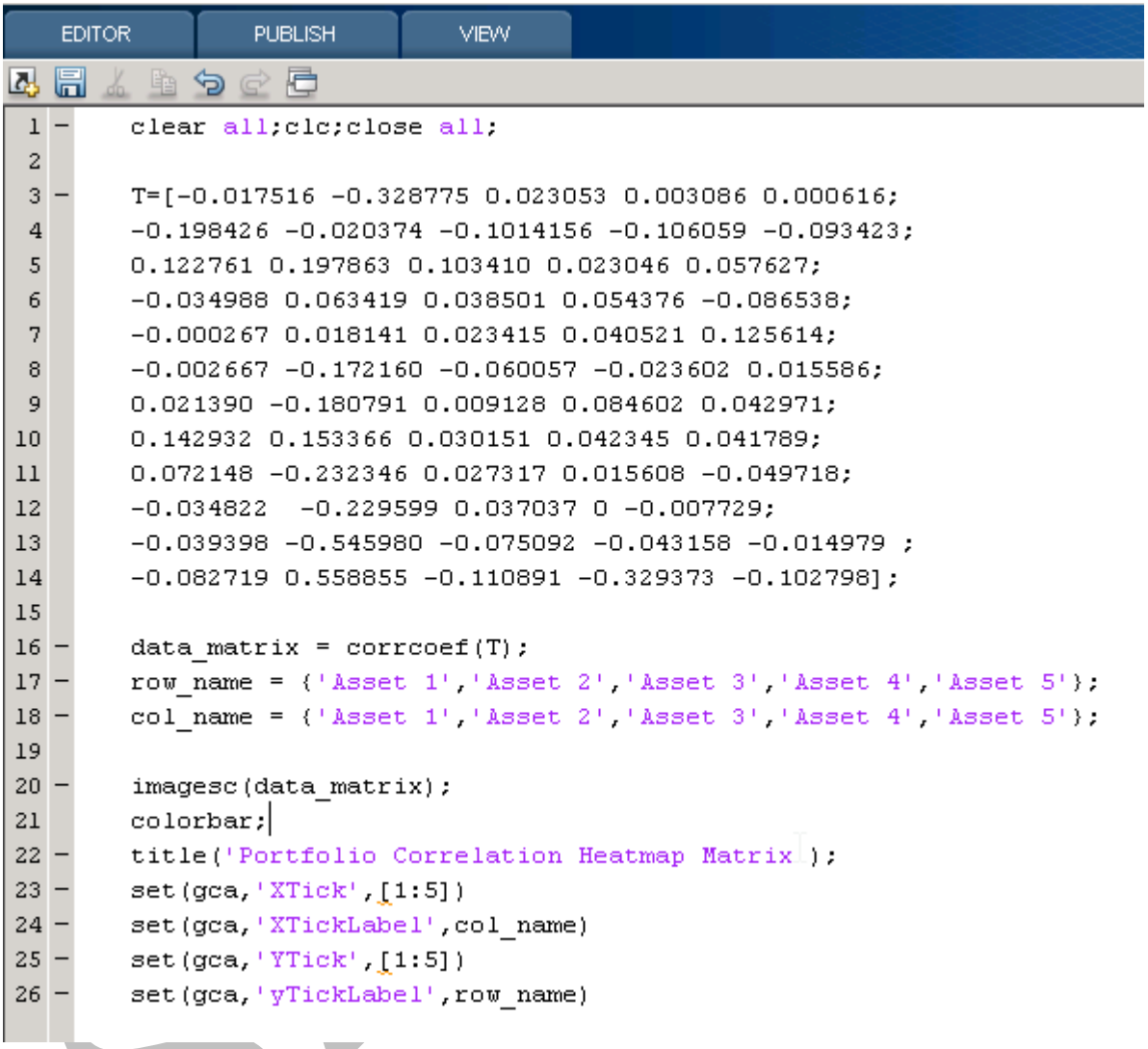


So that's the idea to manage imagery using MATLAB's handle graphics. It's always good practice to save graphics handles to variables if you think you'll need to modify their properties later.

SAMPLE

2D Correlation Heatmap

This is a well know standard in Finance. We have to use a trick of image processing with the function **image( )** to get the result we want but MATLAB is not as flexible as R for this type of job:



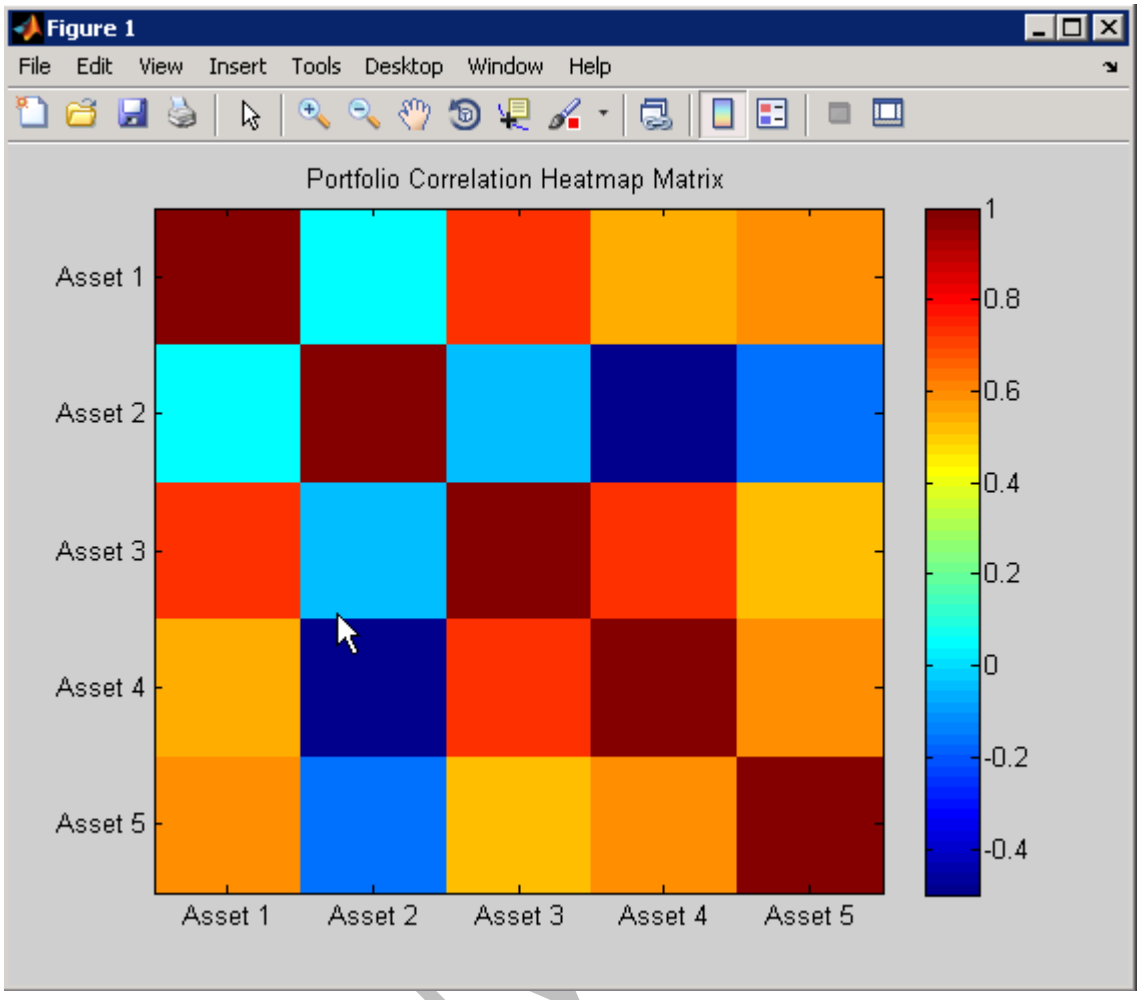

```
clear all;clc;close all;

T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
-0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
0.122761 0.197863 0.103410 0.023046 0.057627;
-0.034988 0.063419 0.038501 0.054376 -0.086538;
-0.000267 0.018141 0.023415 0.040521 0.125614;
-0.002667 -0.172160 -0.060057 -0.023602 0.015586;
0.021390 -0.180791 0.009128 0.084602 0.042971;
0.142932 0.153366 0.030151 0.042345 0.041789;
0.072148 -0.232346 0.027317 0.015608 -0.049718;
-0.034822  -0.229599 0.037037 0 -0.007729;
-0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
-0.082719 0.558855 -0.110891 -0.329373 -0.102798];

data_matrix = corrcoef(T);
row_name = {'Asset 1','Asset 2','Asset 3','Asset 4','Asset 5'};
col_name = {'Asset 1','Asset 2','Asset 3','Asset 4','Asset 5'};

imagesc(data_matrix);
colorbar;
title('Portfolio Correlation Heatmap Matrix');
set(gca,'XTick',[1:5])
set(gca,'XTickLabel',col_name)
set(gca,'YTick',[1:5])
set(gca,'yTickLabel',row_name)
```

The result is on next page:



SAM

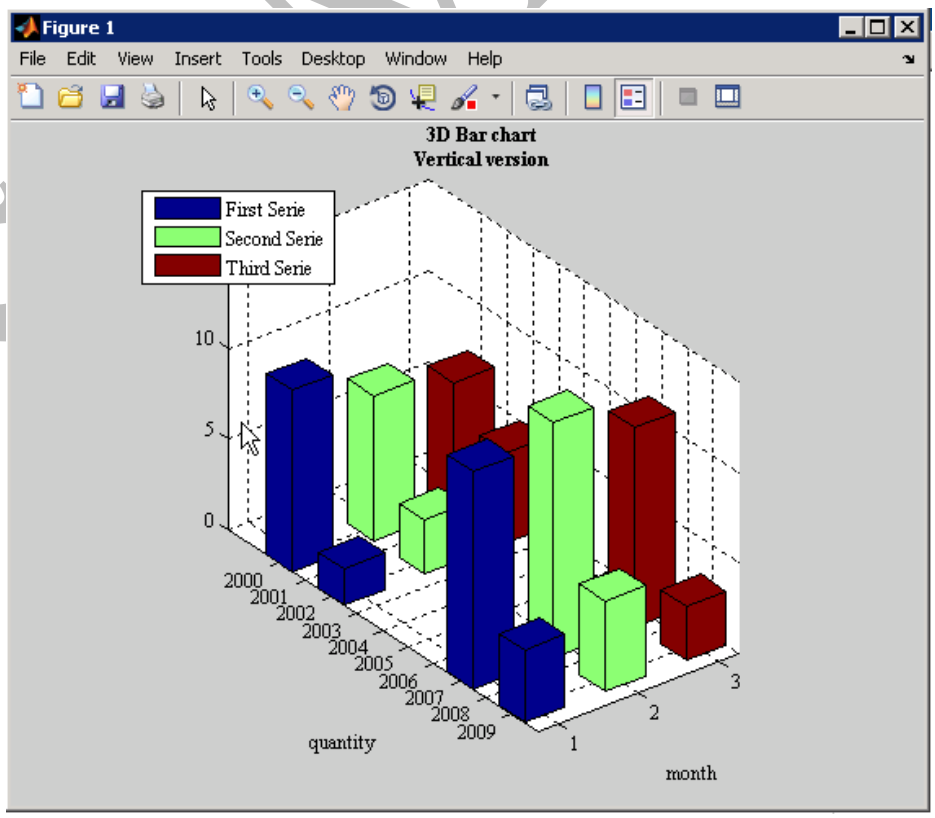
3D vertical bar charts

I'm not a fan of this type of chart then we will go quickly trough. The idea is quite simple but like always, here we use the **bar3( )** function and some tricks to manage ticks correctly with MATLAB (for sure... same can be done horizontally...):

```
1   x=[2 4 9 11];
2   mat=[10 8 7; 2 3 5; 12 13 11; 4 5 3];
3   bar3(x,mat,0.5,'detached');
4   set(gca, 'ytick', [1:1:10]);
5   set(gca, 'yticklabel', (2000:1:2009))
6   set(gca,'FontSize',9,'FontName','Times');
7   title({'3D Bar chart';'Vertical version'},'FontWeight','bold');
8   xlabel('month');
9   ylabel('year');
10  ylabel('quantity');
11  legend('First Serie','Seconde Serie','Third Serie');
12  legend(gca,'location','NorthWest')
    rotate3d on
```

You can see also the rotate3d command to let the user rotate the chart even if no toolbars are available on the chart.

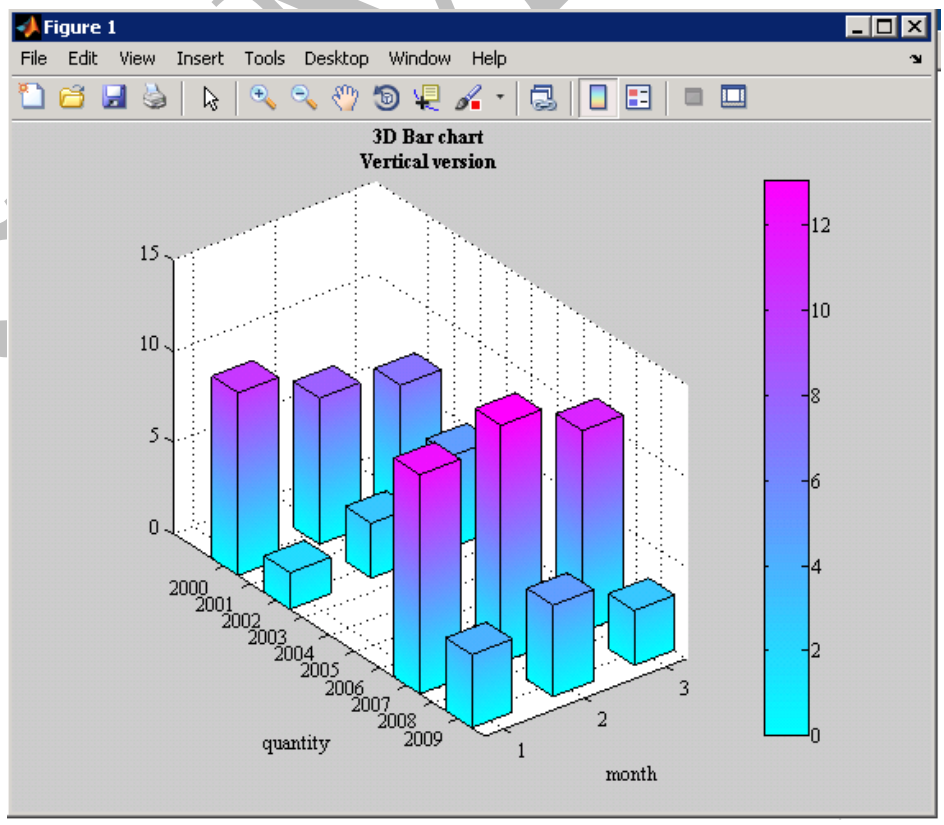
The result will be:



You can make the result a little bit prettier by using a lot of MATLAB hidden tricks:

```
x=[2 4 9 11];
mat=[10 8 7; 2 3 5; 12 13 11; 4 5 3];
h=bar3(x,mat,0.5,'detached');
colormap cool
colorbar
shading interp
for i = 1:length(h)
    zdata = get(h(i),'ZData');
    set(h(i),'CData',zdata)
    % Add back edge color removed by interpolating shading
    set(h,'EdgeColor','k')
end
set(gca, 'ytick', [2:1:11]);
set(gca, 'yticklabel', (2000:1:2009))
set(gca,'FontSize',9,'FontName','Times');
title({'3D Bar chart';'Vertical version'},'FontWeight','bold');
xlabel('month');
ylabel('year');
ylabel('quantity');
```

The result will be:



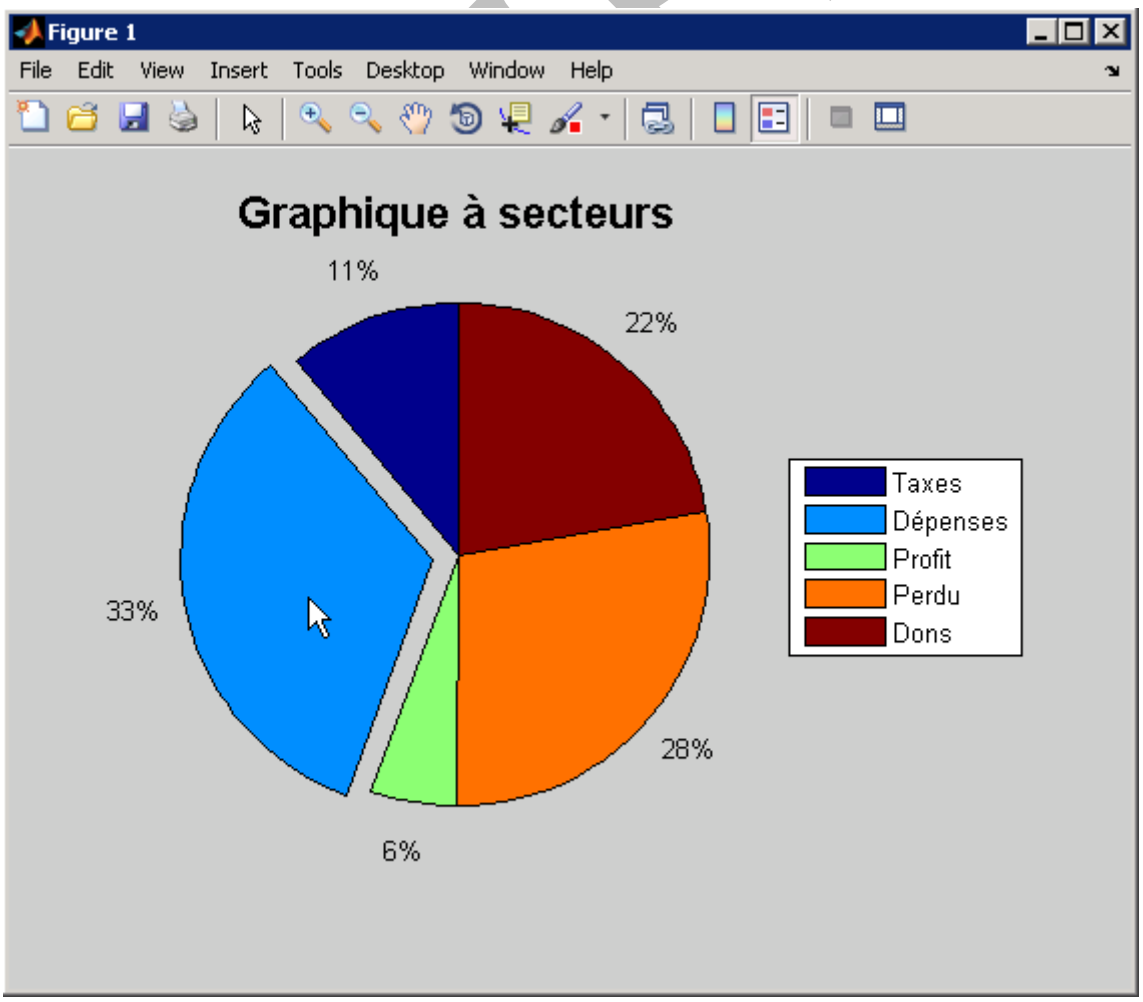
2D/3D sectors charts (cheese/pie charts)

Ok another typical type of chart that is not critical but I strongly recommend you to go on the website of MATLAB file exchange to take a look to the free beautiful 3D sector chart Toolbox.

But here see what are the basics to know first using the function **pie( )** and the trick to add the values to the different sectors:

```
x = [1 3 0.5 2.5 2];
labels = {'Taxes','Dépenses','Profit','Perdu','Dons'};
explode = [0 1 0 0 0];
h=pie(x,explode);
legend(findobj(h, 'type','patch'), labels, 'Location','EastOutside')
colormap jet;
title('Graphique à secteurs','fontweight','bold','fontsize',16);
```

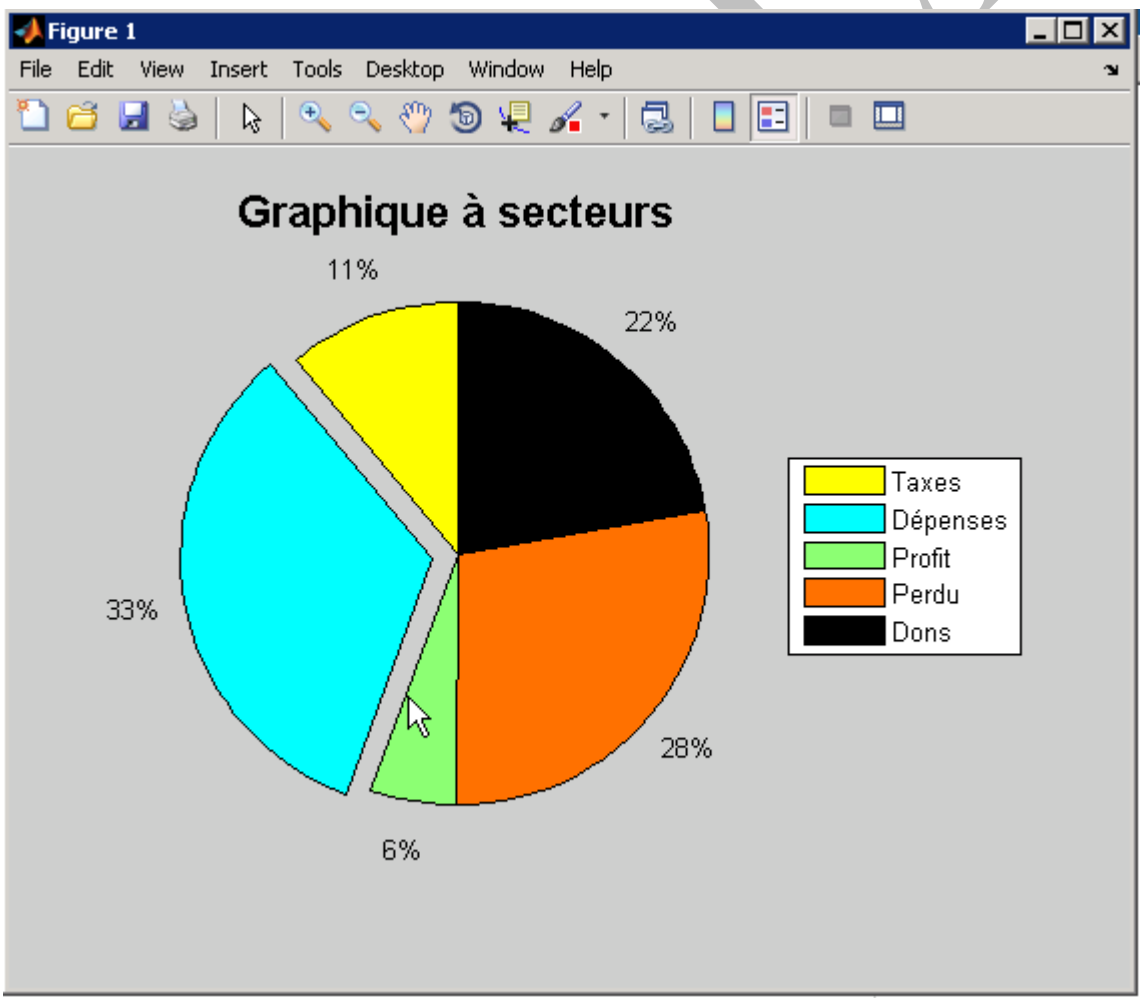
The result is:



Or if we want to control the color for each sector part this is once again difficult to guess:

```
1   clear all;clc;close all
2   x = [1 3 0.5 2.5 2]; %5 values!
3   labels = {'Taxes','Dépenses','Profit','Perdu','Dons'};
4   explode = [0 1 0 0 0];
5   h=pie(x,explode);
6   legend(findobj(h, 'type','patch'), labels, 'Location','EastOutside')
7   %the h vector contains 10 components (2 times 5!)
8   set(h(1),'FaceColor','y') %this is taxis (1-2)
9   set(h(3),'FaceColor','c') %this is Dépenses (3-4)
10  set(h(9),'FaceColor','k') %this is Dons (9-10)
11  title('Graphique à secteurs','fontweight','bold','fontsize',16);
```

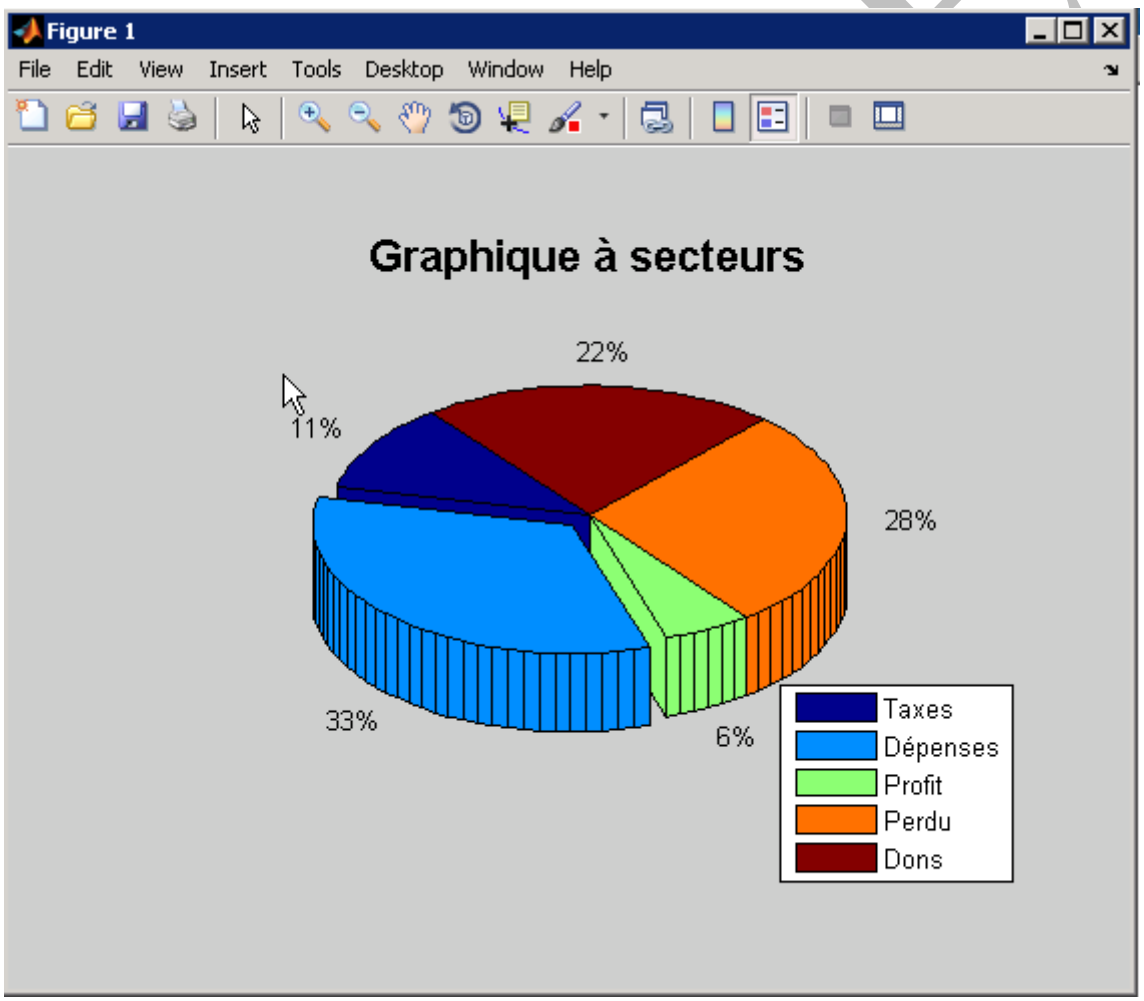
The result is:



And in 3D with the function **pie3( )**:

```
1 -    close all;clear all;clc
2 -    explode=[0 1 0 0 0];
3 -    labels={'Taxes','Dépenses','Profit','Perdu','Dons'};
4 -    x=[1 3 0.5 2.5 2];
5 -    pie3(x,explode);
6 -    legend(labels,'Location','SouthEast');
7 -    title('Graphique à secteurs','fontweight','bold','fontsize',16);
```

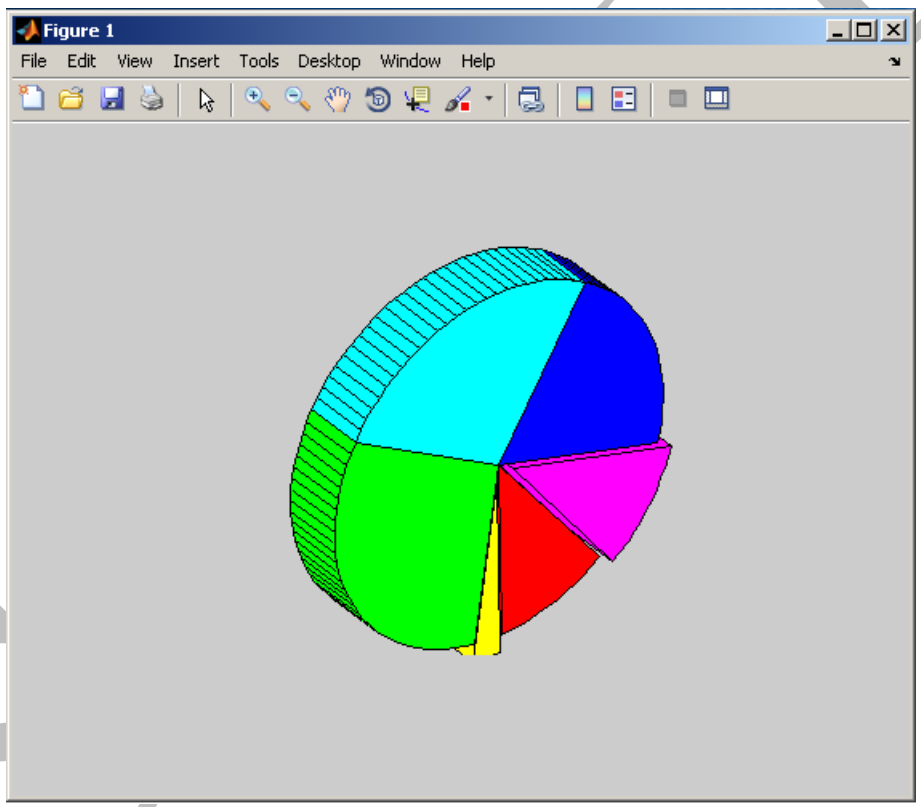
This will give:



We can also create animated 3D pie charts:

```
1   colormap(hsv(6))
2   x = [1 0.25 3 2.5 2 1];
3   explode = [0 1 0 0 0 1];
4   h = pie3(x, explode, repmat({''}, size(x)));
5   n = 180;
6   mov(n) = getframe;
7   for i = 1:n
8       rotate(h, [20 20], 360 / n);
9       mov(i) = getframe;
10  end
11  movie2gif(mov, 'pie4.gif', 'DelayTime', 0.5 / n);
12
```

This give us an animated pie chart:



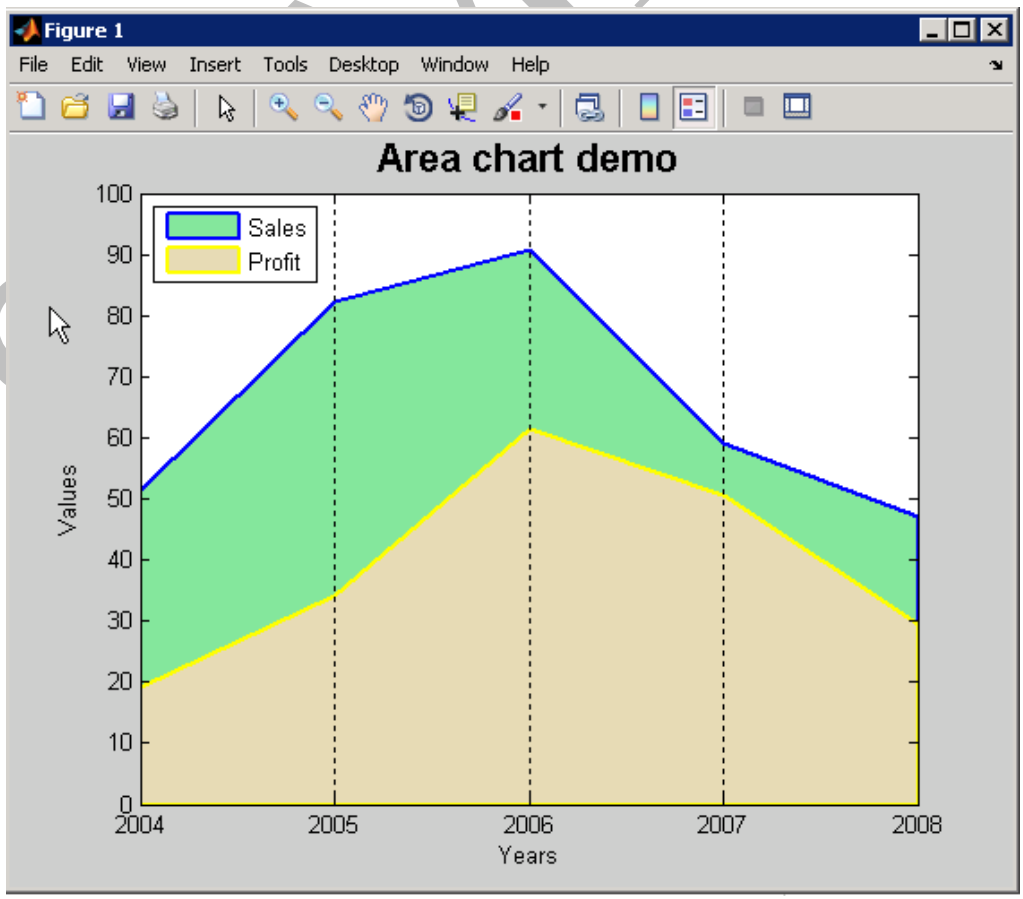
2D Surface/Area charts

Are chart are not used a lot as they are in a first place but they are used a lot in Finance and Econometrics to create custom time series confidence interval (the default one of MATLAB or almost horrible and a too limited).

First let us see a very basic example:

```
clear all;close all;clc
x = 2004:2008;
sales = [51.6 82.4 90.8 59.1 47.0];
profits = [19.3 34.2 61.4 50.5 29.4];
area(x,sales,'FaceColor',[.5 .9 .6],'EdgeColor','b','LineWidth',2);
hold on;
area(x,profits,'FaceColor',[.9 .85 .7],'EdgeColor','y','LineWidth',2);
legend({'Sales','Profit'}, 'Location', 'NorthWest');
set(gca,'XTick',x);
set(gca,'XGrid','on');
set(gca,'Layer','top');
title('Area chart demo','fontweight','bold','fontsize',16);
xlabel('Years');ylabel('Values');
```

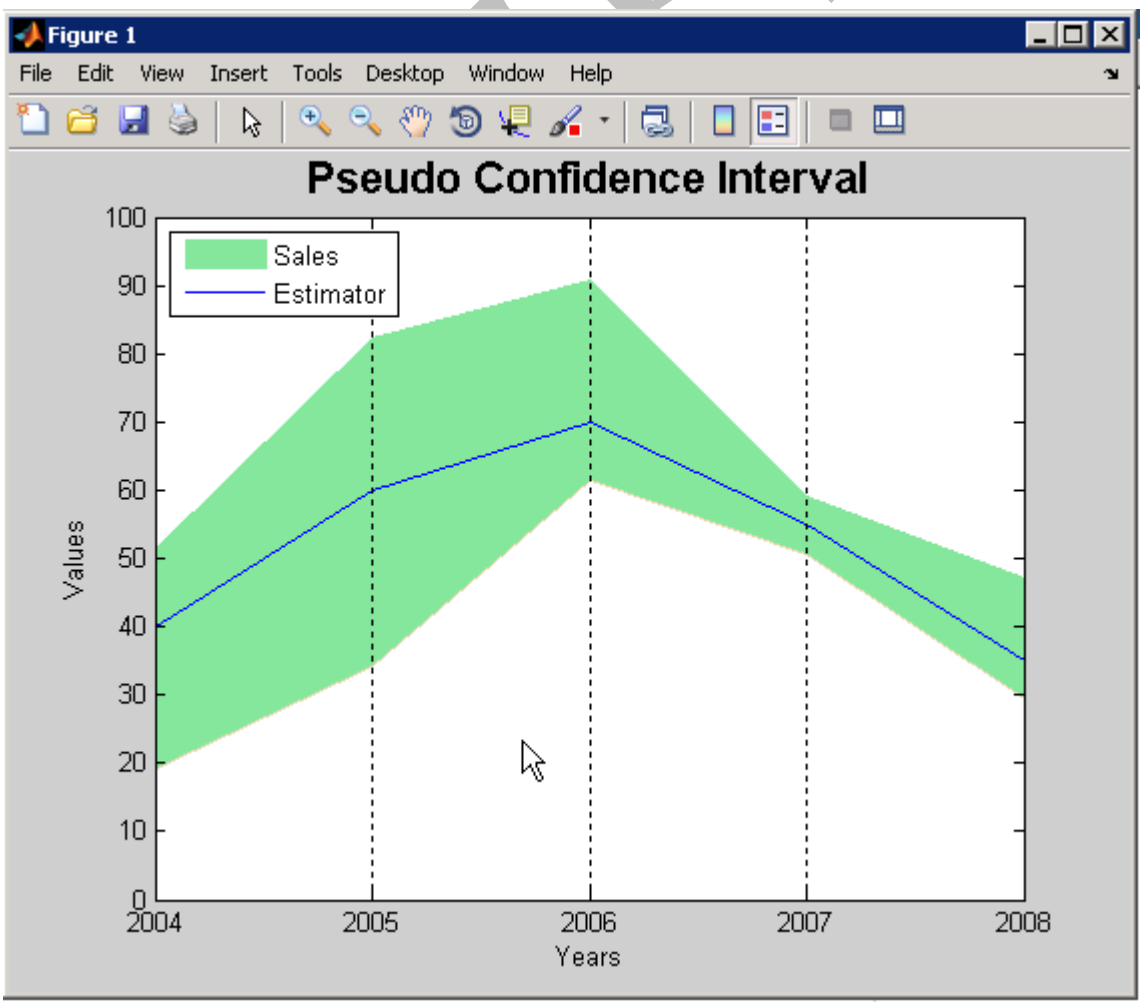
The result will be:



Now a much more interesting usage of area charts is to use them as pseudo-confidence interval. Here is a small example of the trick that mixed the function **area( )** and **plot( )** that you can complicate much more with multiple confidence intervals not just one!

```
x = 2004:2008;
sales = [51.6 82.4 90.8 59.1 47.0];
profits = [19.3 34.2 61.4 50.5 29.4];
forecast = [40 60 70 55 35];
h1=area(x,sales,'FaceColor',[.5 .9 .6],'EdgeColor',[.5 .9 .6]);
hold on;
h2=area(x,profits,'FaceColor',[1 1 1],'EdgeColor',[.9 .85 .7]);
h3=plot(x,forecast);
legend([h1 h3],{'Sales','Estimator'}, 'Location', 'NorthWest');
set(gca,'XTick',x);
set(gca,'XGrid','on');
set(gca,'Layer','top');
title('Pseudo Confidence Interval','fontweight','bold','fontsize',16);
xlabel('Years');ylabel('Values');
```

The result will be:



Radial (rose) bar chart/arrow (compass) chart

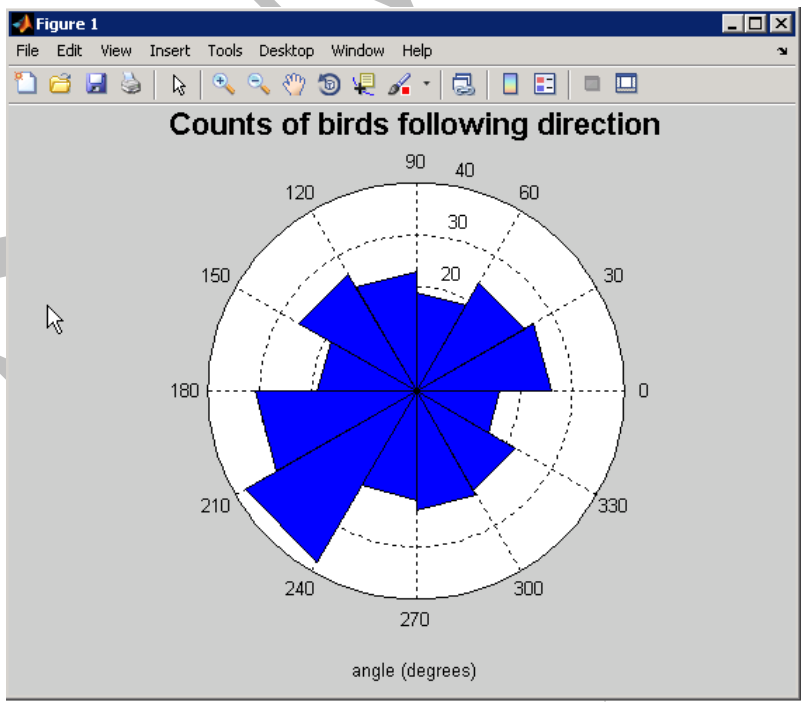
The radial bar chart is especially used in spatial statistics and electronic.

Caution! Even if the **rose( )** function give as output a chart in degrees, the input vector must be in radians!

Even if the chat below is a simple example, it is false because we did not convert the angles in radians... but that doesn't really matter because the purpose is just to see the syntax of the function **rose( )**:

```
clear all;close all;clc
load sunspot.dat
%we take the second column of the dataset that contains the counts
theta = sunspot(:,2);
%we divide by 12 bins (360°/12=30°)
rose_plot=rose(theta,12);
title('Counts of birds following direction','fontweight',...
    'bold','fontsize',16)
xlabel('angle (degrees)');
%we change the colors of the bars
x = get(rose_plot,'XData') ;
y = get(rose_plot,'YData') ;
rose_plot = patch(x, y, 'b');
```

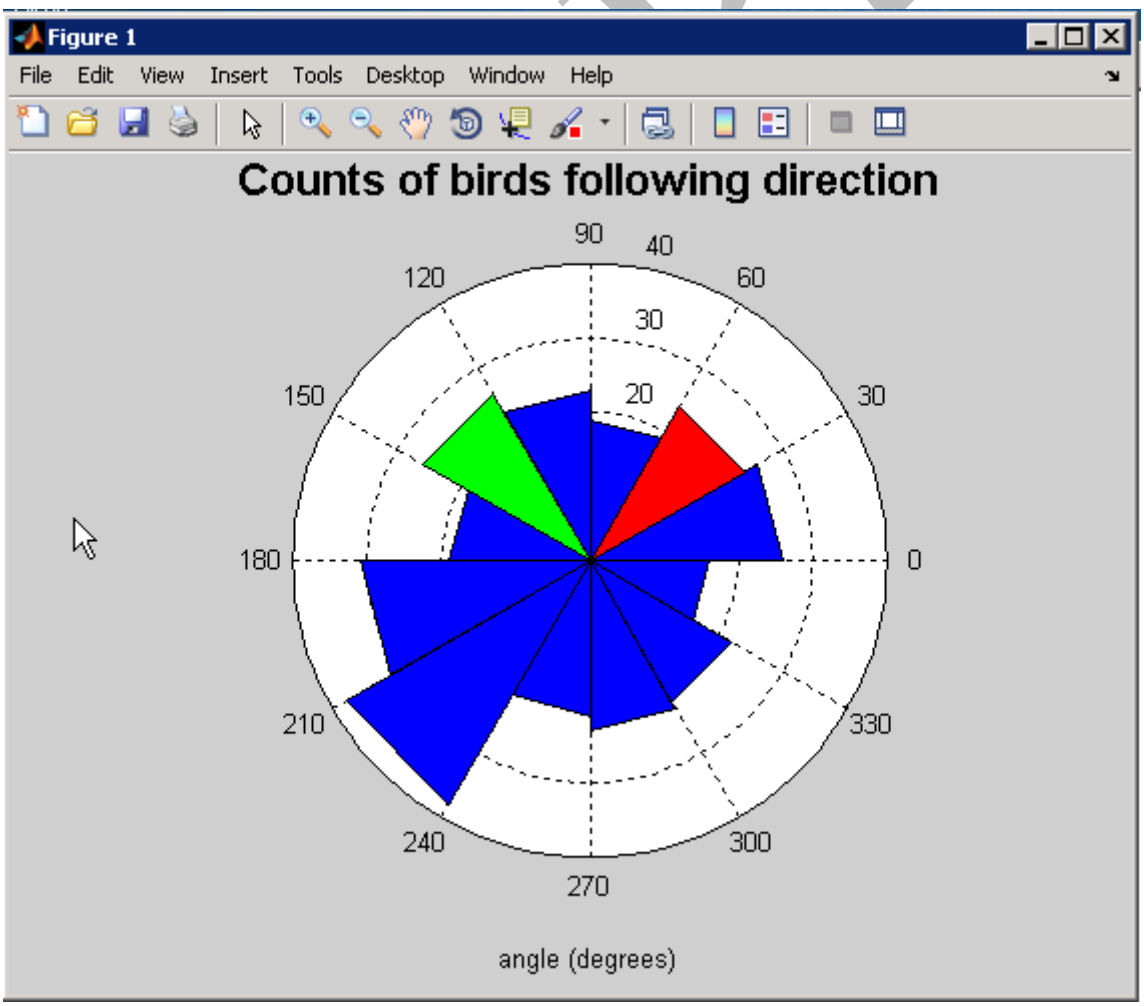
The result is:



And because each vector has 48 components that mean that each triangle is in reality a rectangle with 4 coordinates. Thus to change the color of only one rectangle we have to play a little bit:

```
1   load sunspot.dat
2   theta=sunspot(:,2);
3   rose_plot=rose(theta,12);
4   title('Counts of birds following direction','fontweight',...
5       'bold','fontsize',16)
6   xlabel('angle (degrees)')
7   x=get(rose_plot,'XData');
8   y=get(rose_plot,'YData');
9   rose_plot=patch(x,y,'b');
10  rose_plot=patch(x(5:9),y(5:9),'r');
11  rose_plot=patch(x(16:20),y(16:20),'g');
```

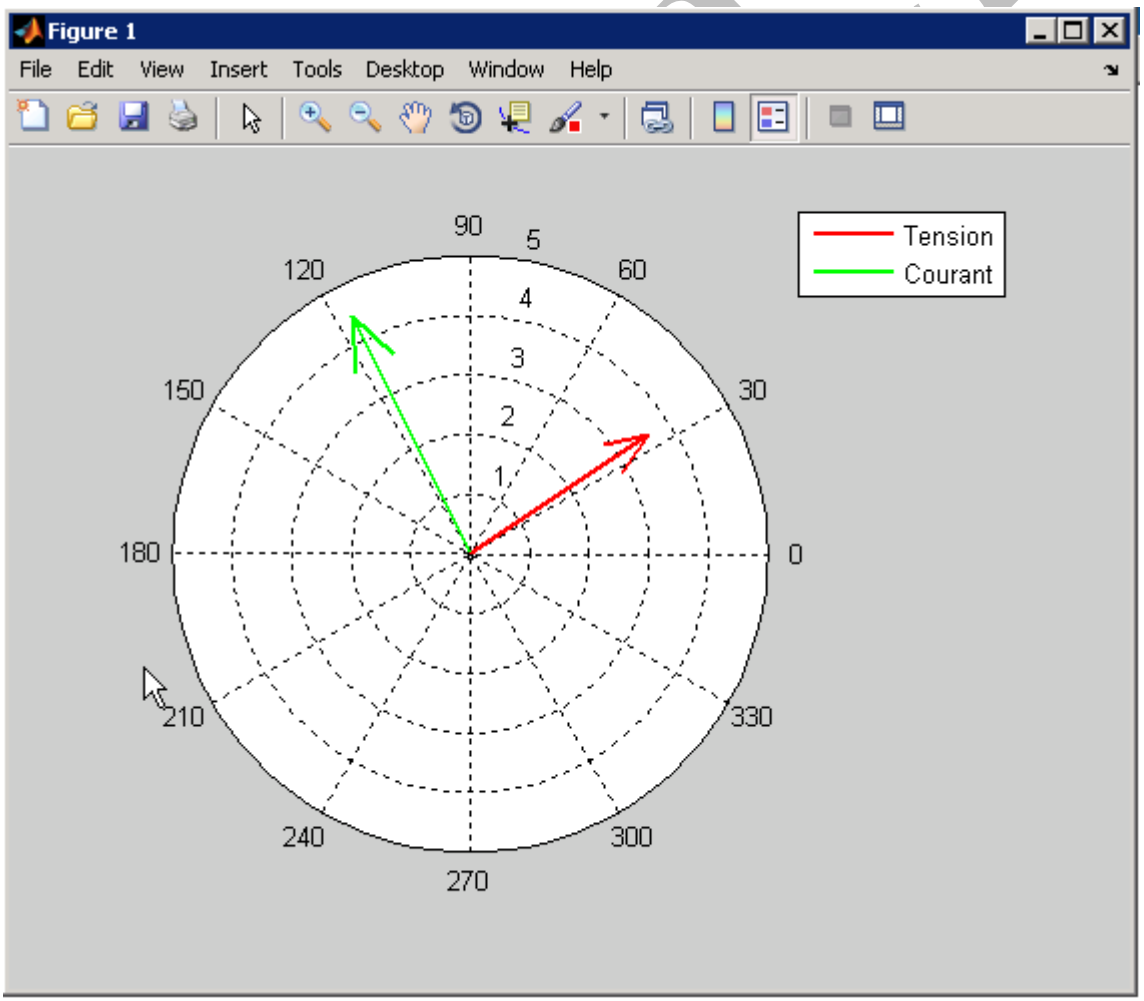
The result will be:



And now for electronic engineers see how we can plot phase vectors using the **compass( )** function and some tricks to change the color of the arrows...:

```
close all;clear all;clc
Z=[3+2*1i;-2+4*1i];
color = [1 0 0;0 1 0]; %red and green
h=compass(Z);
set(h,{'Color'},num2cell(color,2),'LineWidth',2);
str={'Tension' 'Courant'};
legend(h,str,'Location','BestOutside');
```

The result will be:



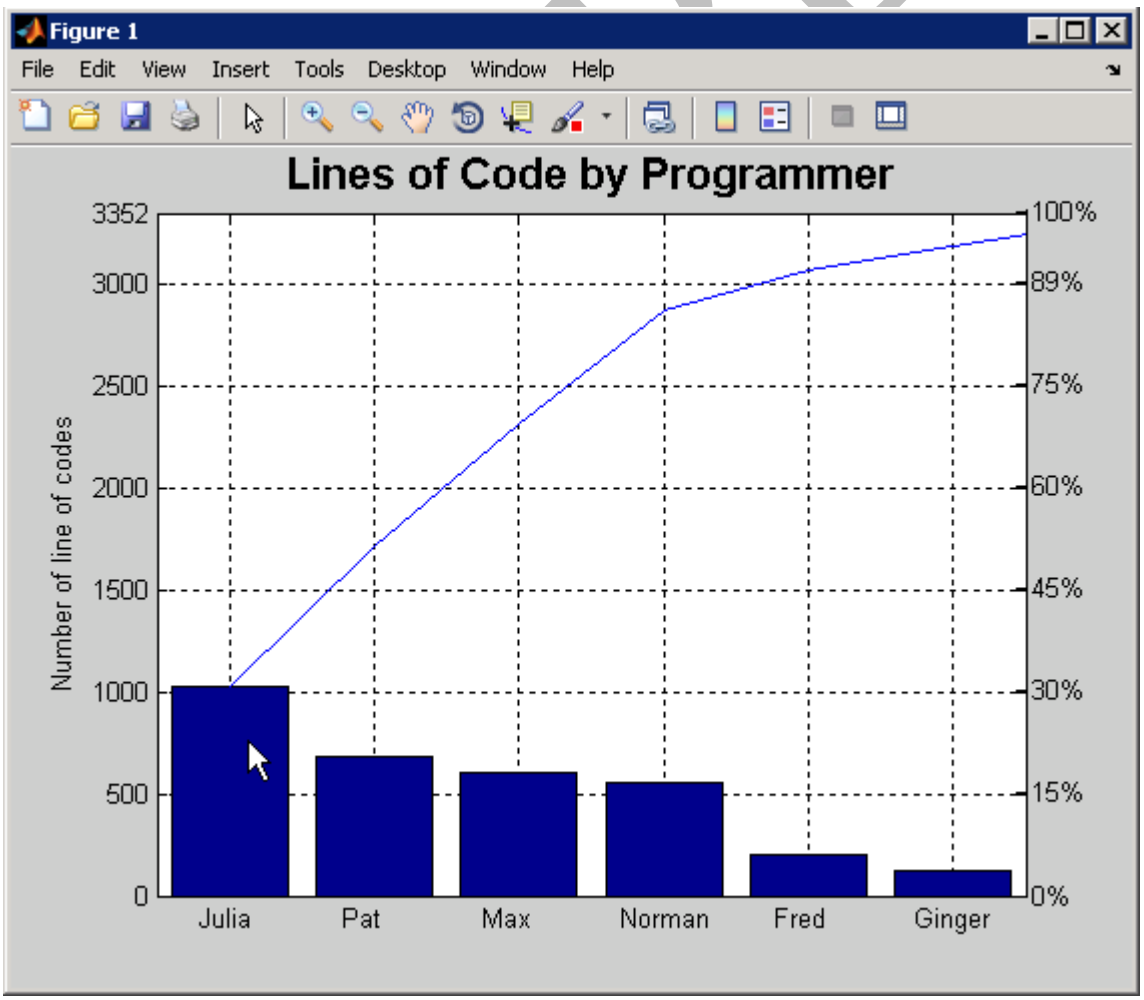
Pareto chart

Pareto charts in MATLAB are very limited and basics as in most Statistical softwares by default. Then you will have to do a lot of additions to get something professional.

But let us see the default possibilities of MATLAB about this type of chart using the function **pareto( )**:

```
1   codelines = [200 120 555 608 1024 101 57 687];
2   coders = {'Fred','Ginger','Norman','Max','Julia','Wally','Heidi','Pat'};
3   pareto(codelines, coders)
4   title('Lines of Code by Programmer','fontweight',...
5       'bold','fontsize',16)
6   ylabel('Number of line of codes');
7   grid on
```

And the result will be...:



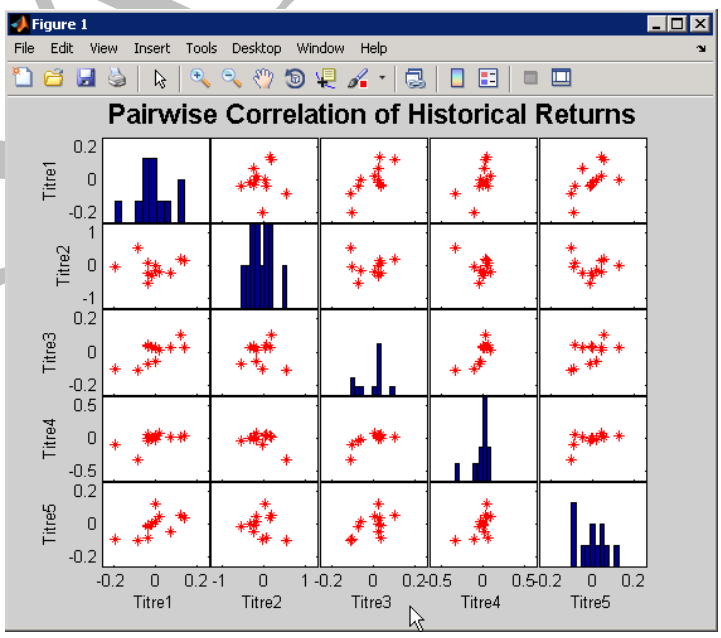
Plotmatrix

Matrix plots are very disappointing in MATLAB in comparison of R... The only one thing that we can do (as far as I know) are histograms on the diagonal and dot charts otherwise.

Let us see this with our famous small dataset of 5 assets and by using the function **plotmatrix( )**:

```
T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
-0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
0.122761 0.197863 0.103410 0.023046 0.057627;
-0.034988 0.063419 0.038501 0.054376 -0.086538;
-0.000267 0.018141 0.023415 0.040521 0.125614;
-0.002667 -0.172160 -0.060057 -0.023602 0.015586;
0.021390 -0.180791 0.009128 0.084602 0.042971;
0.142932 0.153366 0.030151 0.042345 0.041789;
0.072148 -0.232346 0.027317 0.015608 -0.049718;
-0.034822  -0.229599 0.037037 0 -0.007729;
-0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
-0.082719 0.558855 -0.110891 -0.329373 -0.102798];
[~,ax]=plotmatrix(T,'*r');
title('Pairwise Correlation of Historical Returns','fontweight',...
    'bold','fontsize',16);
names={'Titre1', 'Titre2', 'Titre3', 'Titre4', 'Titre5'};
for i = 1:length(names)
    ylabel(ax(i,1),names{i});
    xlabel(ax(end,i),names{i});
end
```

The result will be:



3D parametric plots

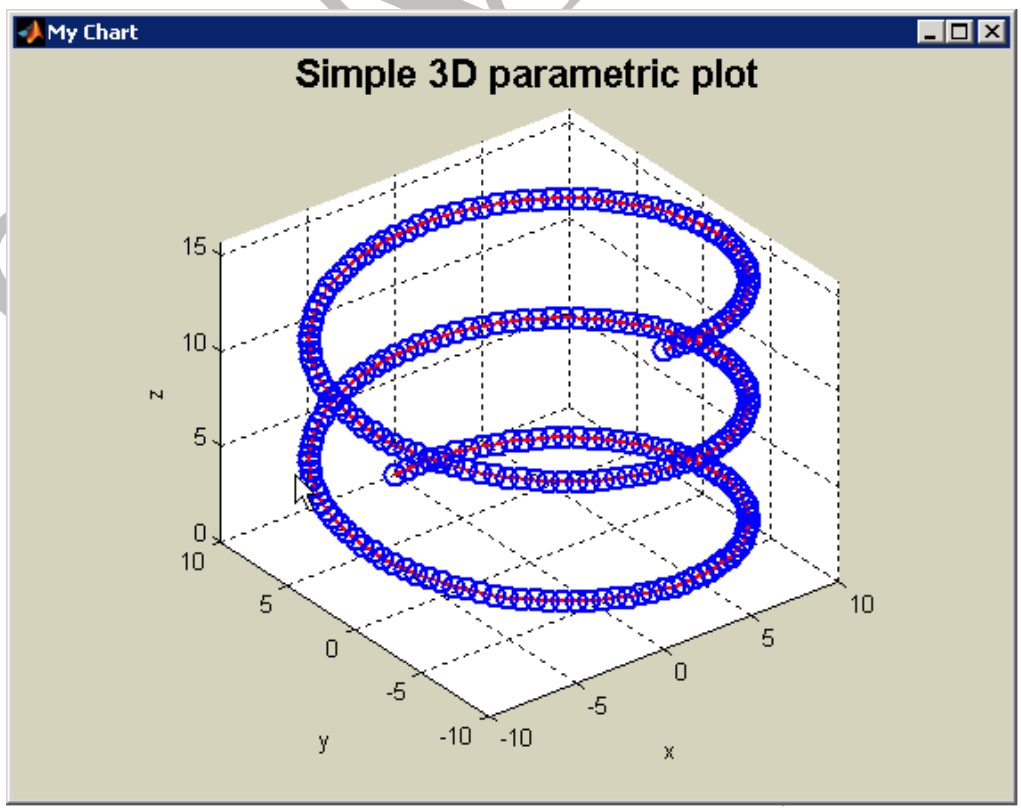
3D parametric plots are especially used by engineers that work on spatial rockets or with drones. We can see also some biologists plotting the trajectory of a tracked bird. Otherwise it is more a plot family use in academics fields for pedagogical purposes.

Vector based 3D parametric line plot

Let us begin with the function **plot3( )** that can be useful to plot a parametric line in space with or without points visible on the line when you have vector of points in the time:

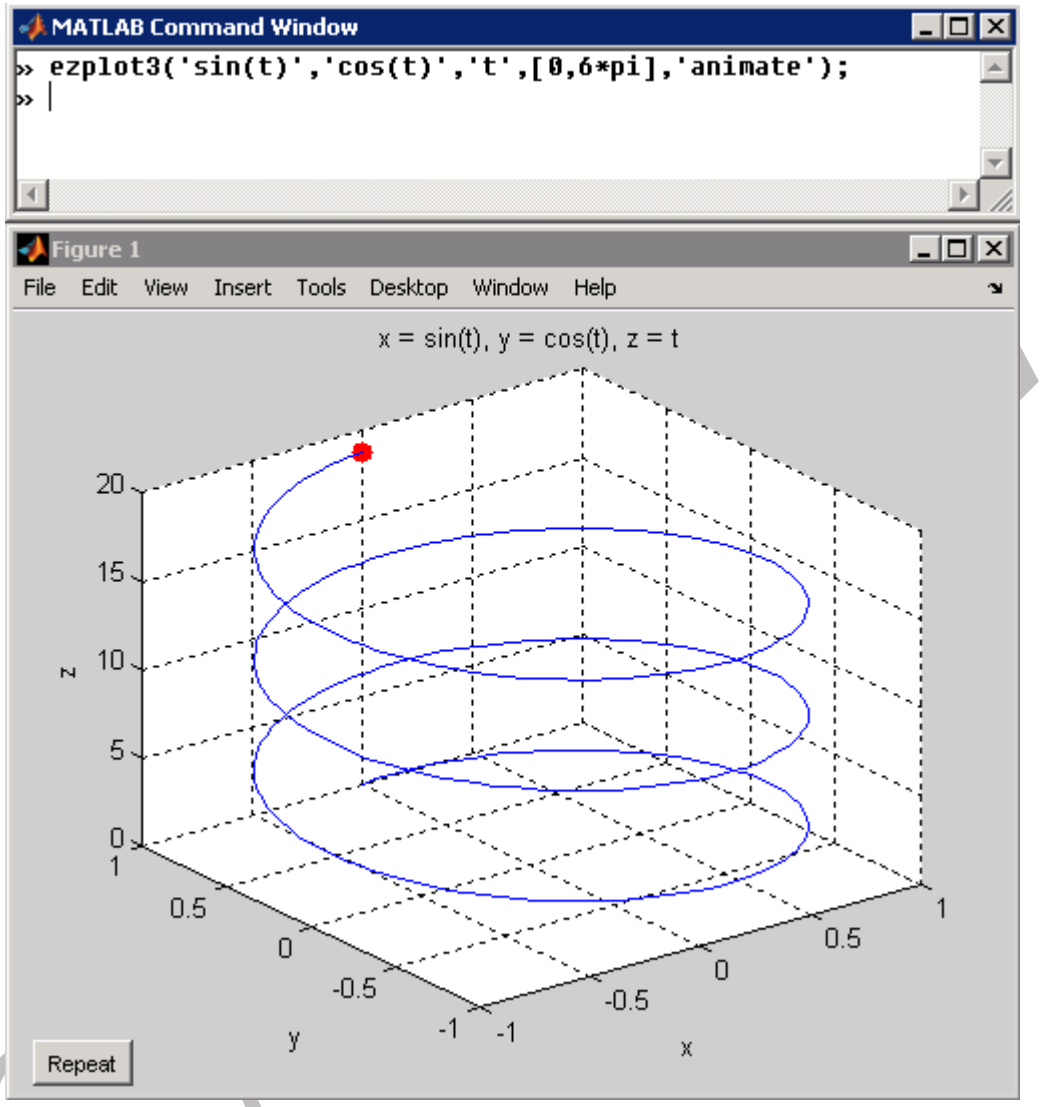
```
1   clear all;close all;clc
2   figure;
3   set(gcf,'Menubar','none','Name','My Chart','NumberTitle','off',...
4    'Color',[0.8314 0.8157 0.7483]);
5   t = 0:pi/50:5*pi;
6   st = 10*sin(t);
7   ct = 10*cos(t);
8   plot3(st,ct,t,'-ro','LineWidth',2,'Markersize',10,'MarkerEdgeColor','b')
9   grid on;
10  xlabel('x');ylabel('y');zlabel('z')
11  title('Simple 3D parametric plot','fontweight','bold','fontsize',16);
12  axis equal;
```

The result will be:



Animated or non-animated analytic based 3D parametric line plot

If you want to animate an analytical parametric plot expression (typical in academics classes) you can easily do it by using the function **ezplot3( )**:



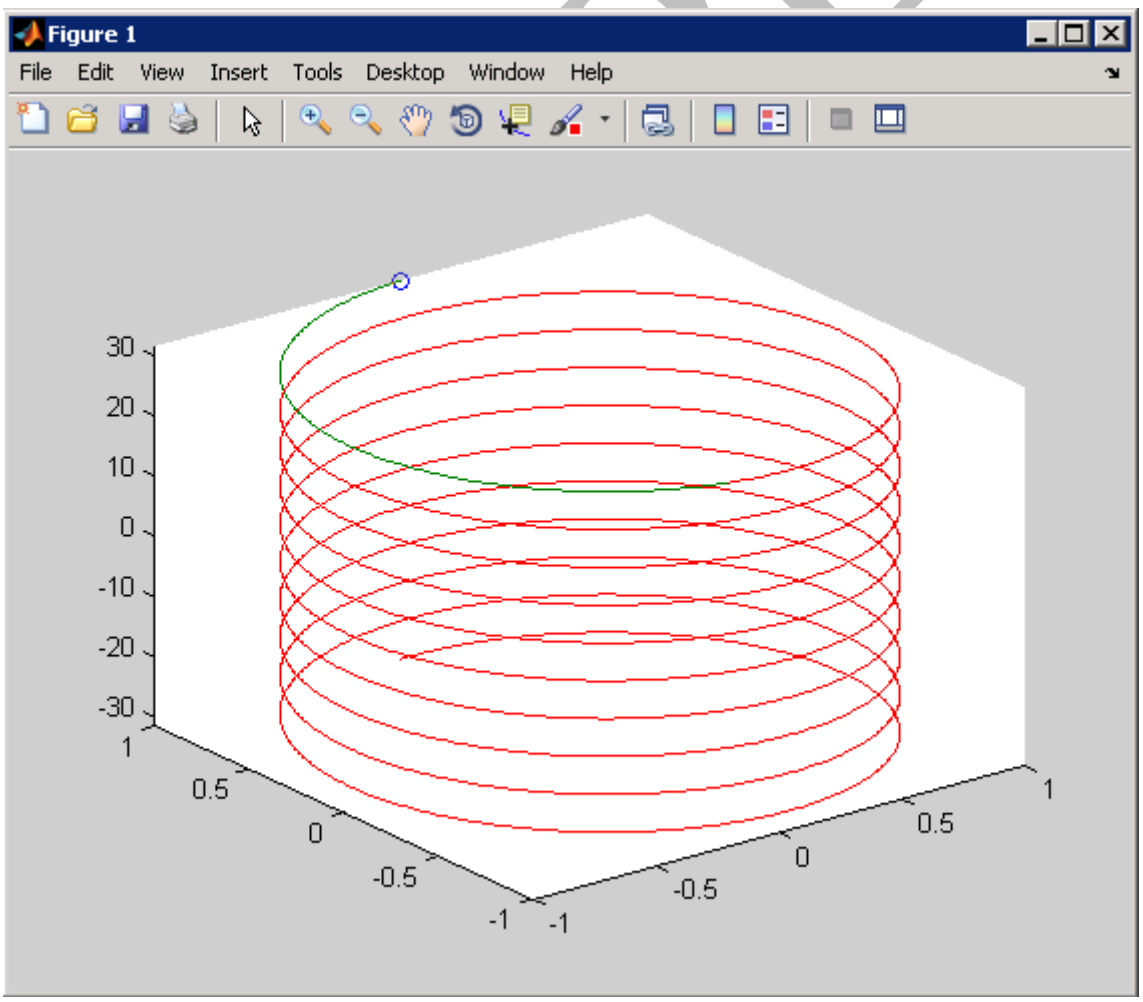
See the **Repeat** button on the bottom left corner!

Animated vector based 3D parametric line plot

If you don't have an analytical expression but only vector with the points of evolution in space-time the **comet3( )** function can show you the animated plot by doing a linear interpolation (when possible...) between each for the fluidity of the animation:

```
%same thing with comet3 but it use a linear interpolation
%and we can see that by increasing the step
t = -10*pi:pi/250:10*pi;
comet3(sin(t),cos(t),t);
%axis and title will appear after animation
xlabel('x');ylabel('y');zlabel('z')
title('x=sin(t),y=cos(t),z=t');
```

The result will be:

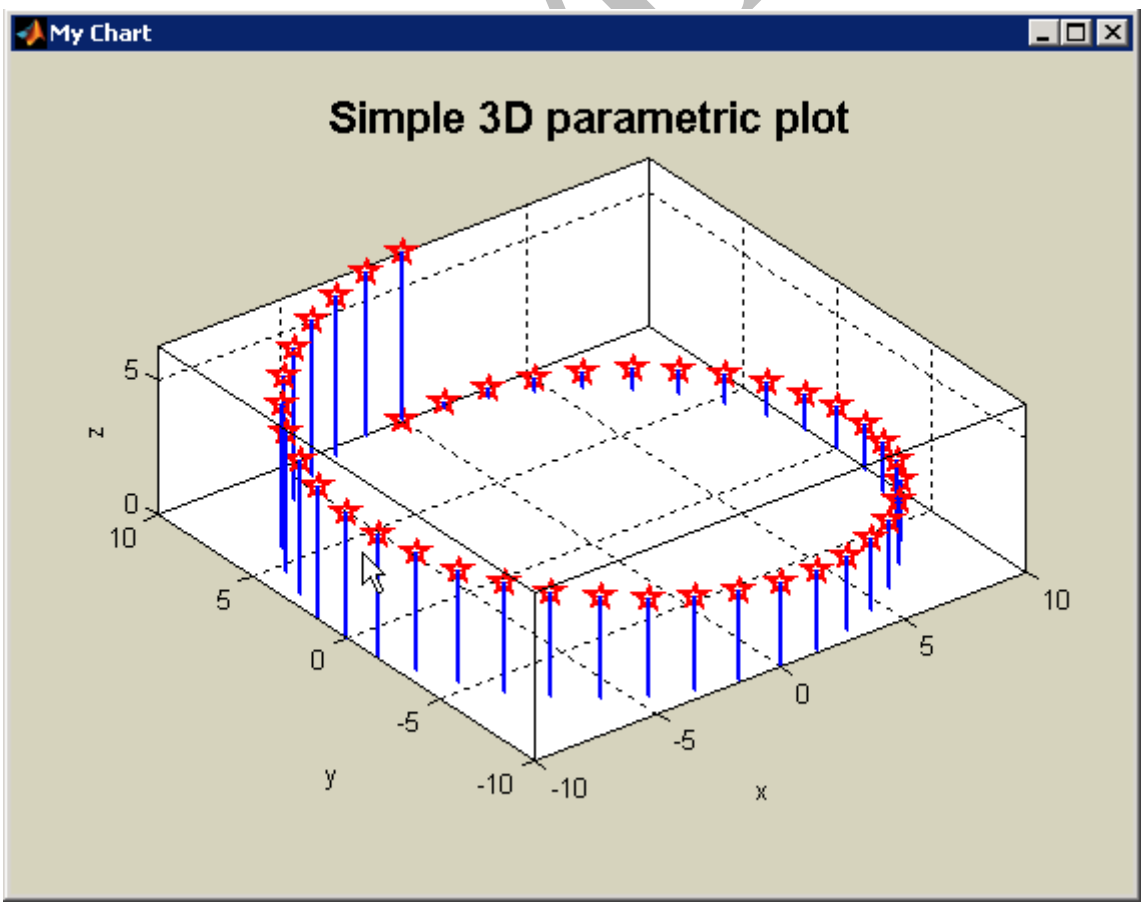


Vector based 3D stem parametric line plot

This plot is especially used in rocket science to see over what the rocket was on the ground (or on the water in most cases...) on in GPS tracking. For this purpose you use the function **stem3( )**:

```
clear all;close all;clc
figure;
set(gcf,'Menubar','none','Name','My Chart','NumberTitle','off',...
 'Color',[0.8314 0.8157 0.7483]);
t = 0:pi/20:2*pi;
st = 10*sin(t);
ct = 10*cos(t);
stem3(st,ct,t,'bp','LineWidth',2,'Markersize',10,'MarkerEdgeColor','r')
grid on;
xlabel('x');ylabel('y');zlabel('z')
title('Simple 3D parametric plot','fontweight','bold','fontsize',16);
axis equal;
```

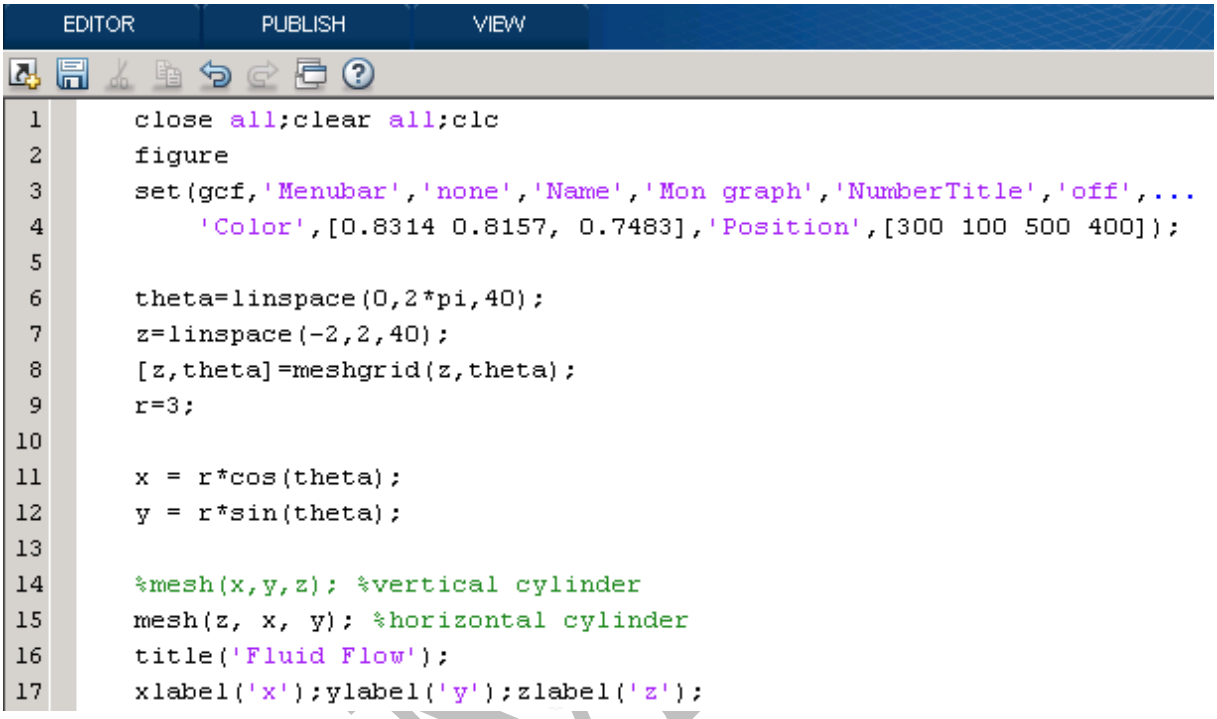
The result will look like this (**ToDo**: I still have to add an image on the ground when I will have the time):



Vector based 3D surface parametric plot

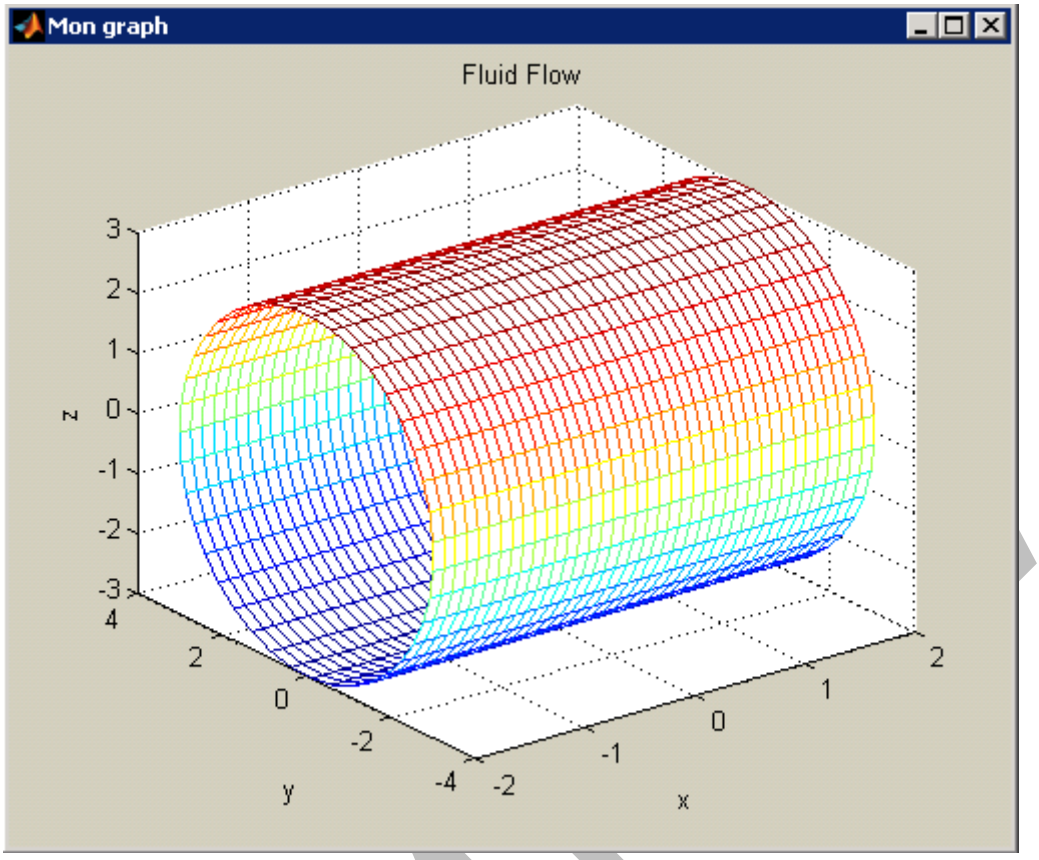
This type of plots is mainly academics but I saw them sometimes used in General Relativity (but still for academic pedagogical purposes).

Let us begin with the most simple parametric plot (a cylinder) to deeply understand how to manipulate the **meshgrid( )** and **mesh( )** functions that are also sometimes used in Finance:



That will give:

SAMPLE

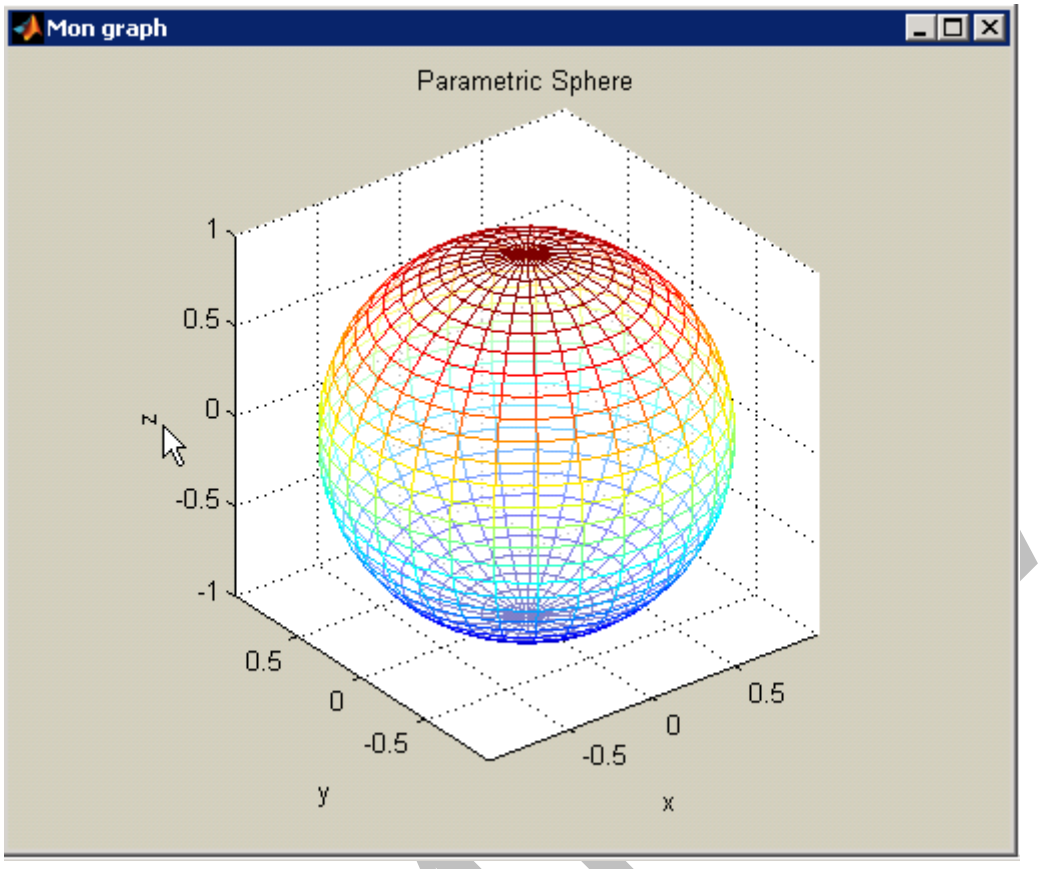


And what about a simple sphere with transparency on the surface to be able to see trough:

EDITOR PUBLISH VIEW

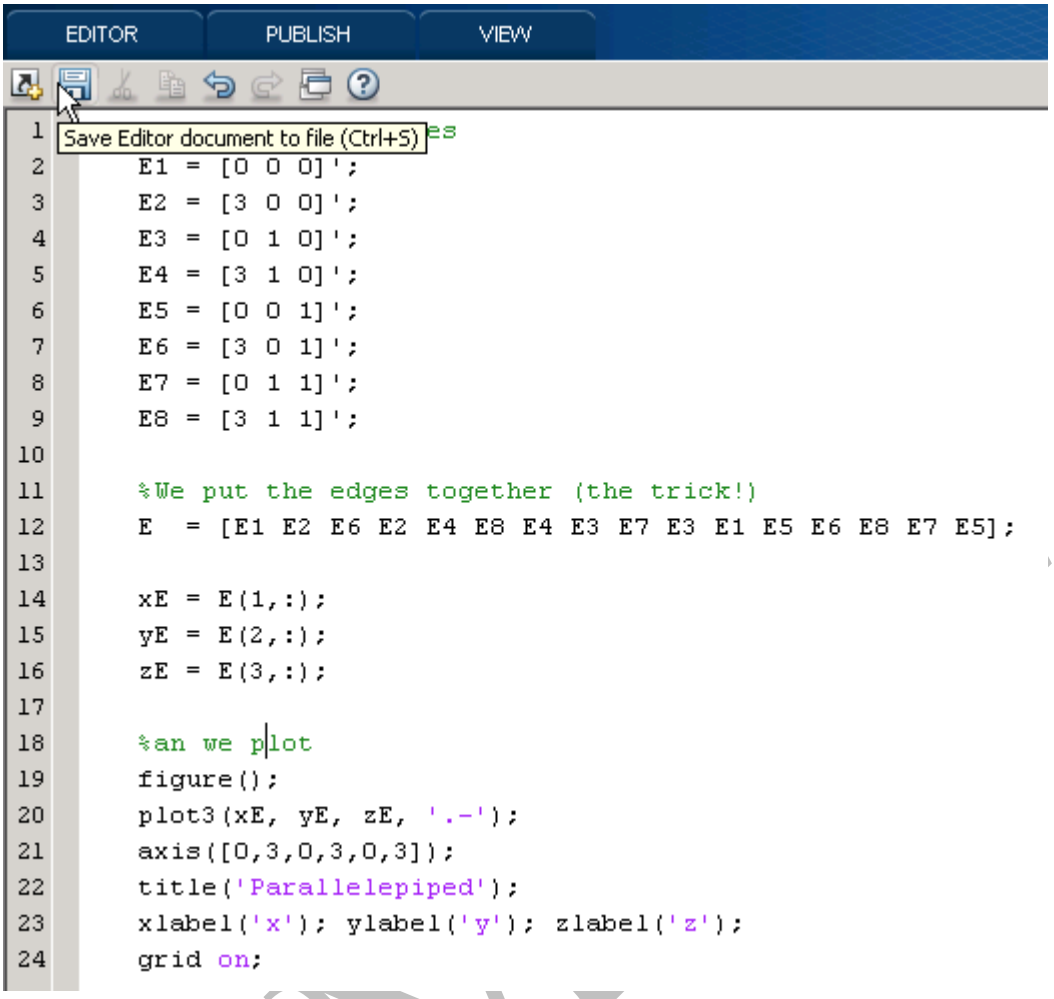
```
1 -    close all;clear all;clc
2 -    figure
3 -    set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4          'Color',[0.8314 0.8157, 0.7483],'Position',[300 100 500 400]);
5
6 -    phi=linspace(0,pi,30);
7 -    theta=linspace(0,2*pi,30);
8 -    [phi,theta]=meshgrid(phi,theta);
9
10 -   x=sin(phi).*cos(theta);
11 -   y=sin(phi).*sin(theta);
12 -   z=cos(phi);
13
14 -   mhndl=mesh(x,y,z);
15 -   set(mhndl,'FaceAlpha',0.5);
16 -   axis equal;
17
18 -   title('Parametric Sphere');
19 -   xlabel('x');ylabel('y');zlabel('z');
```

The result will be:



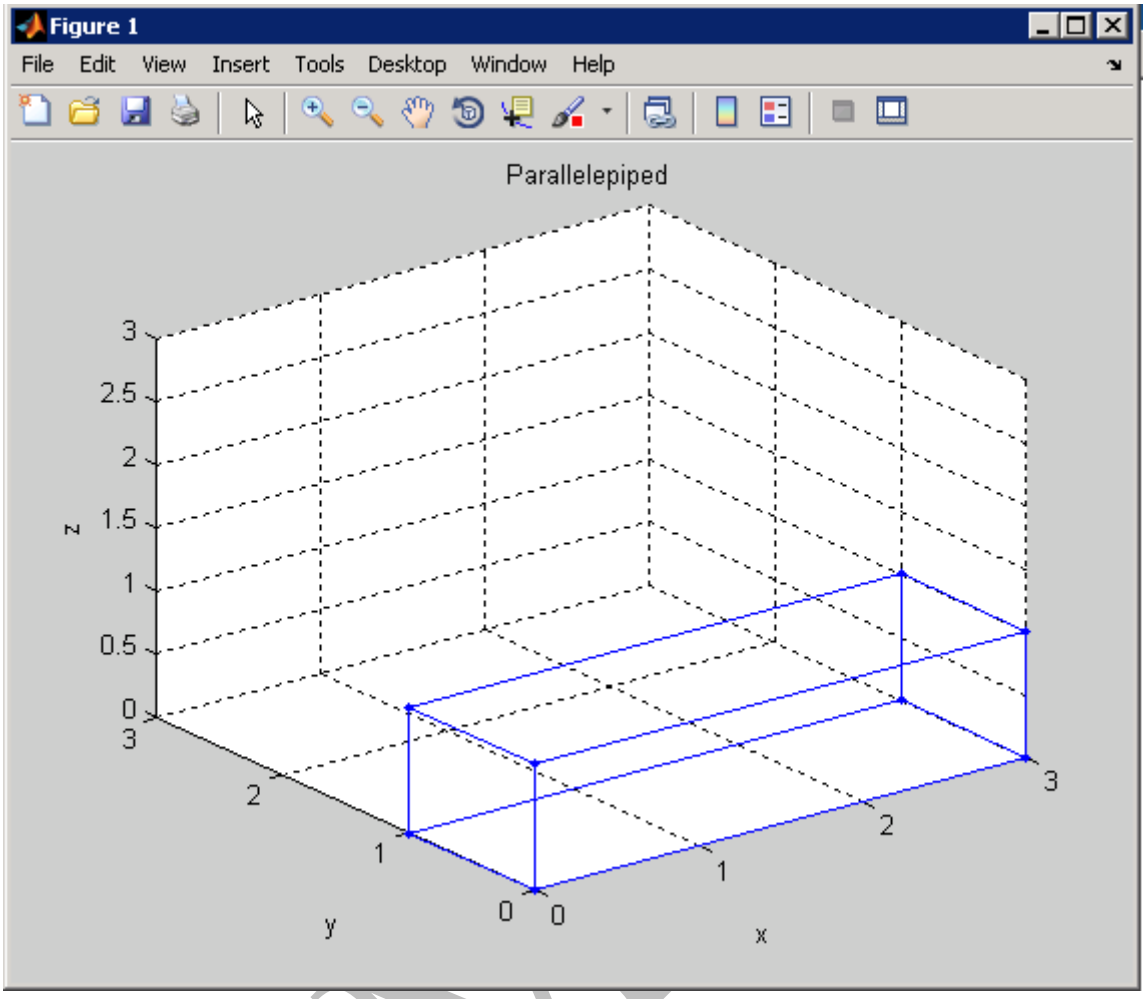
Something that seems to be easy but has no built-in function in MATLAB as far as I know is to create a parallelepiped (useful for physicists and chemists because a lot of probes have this shape):

SAMPLE



This will give:

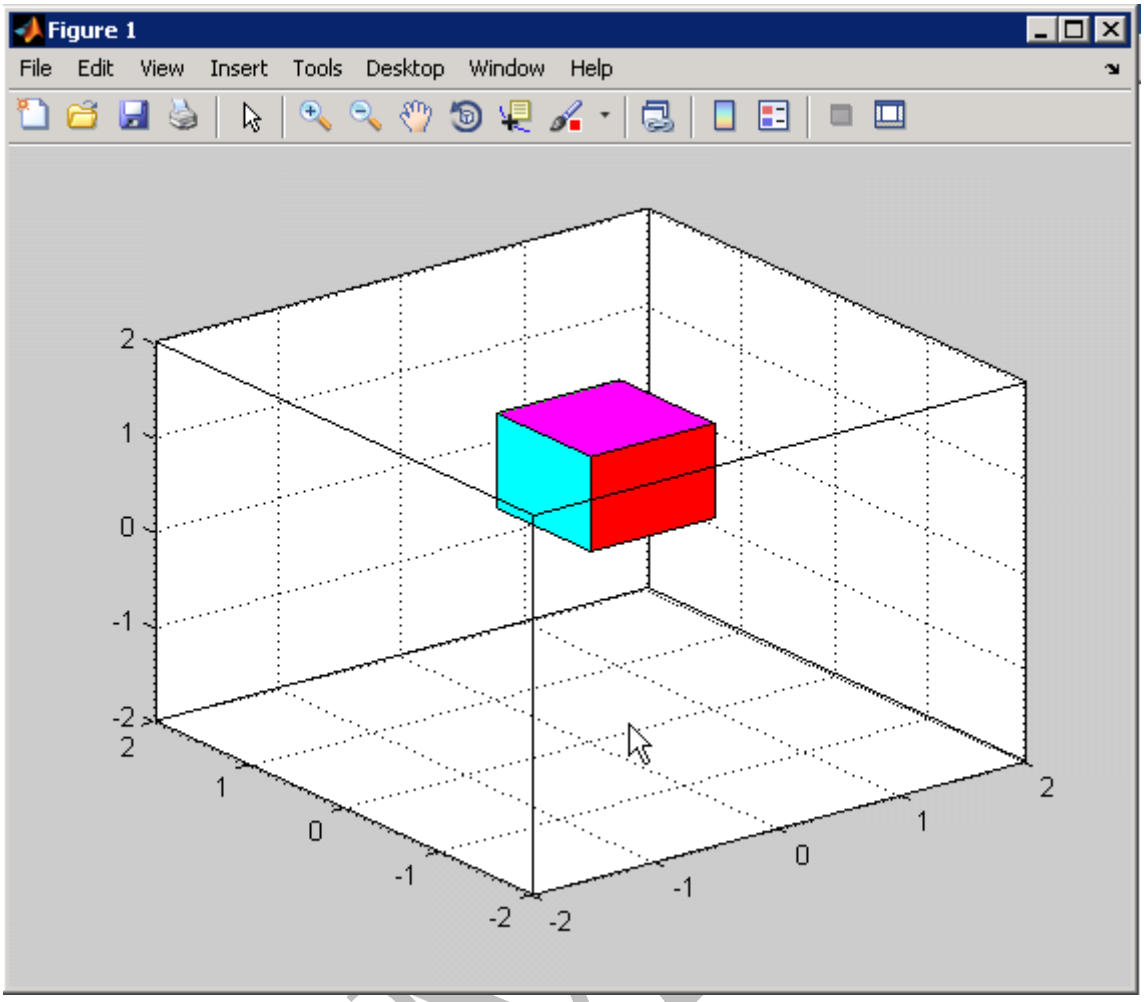
SAM



Or in another way using the function **patch( )**:

```
1     %vector with the 8 points of the parallelepiped
2 -   vm=[0 0 0; 1 0 0; 1 1 0; 0 1 0; 0 0 1; 1 0 1; 1 1 1; 0 1 1];
3
4     %because vm must be a row vector for each of the 6 faces we will reference
5     %the four building points of vm
6 -   fm=[1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 1 2 3 4; 5 6 7 8];
7
8 -   patch('Vertices',vm,'Faces',fm, 'FaceVertexCData',hsv(6),'FaceColor','flat');
9 -   axis([-2 2 -2 2 -2 2]);
10 -  grid on;box on;
```

The result will be:

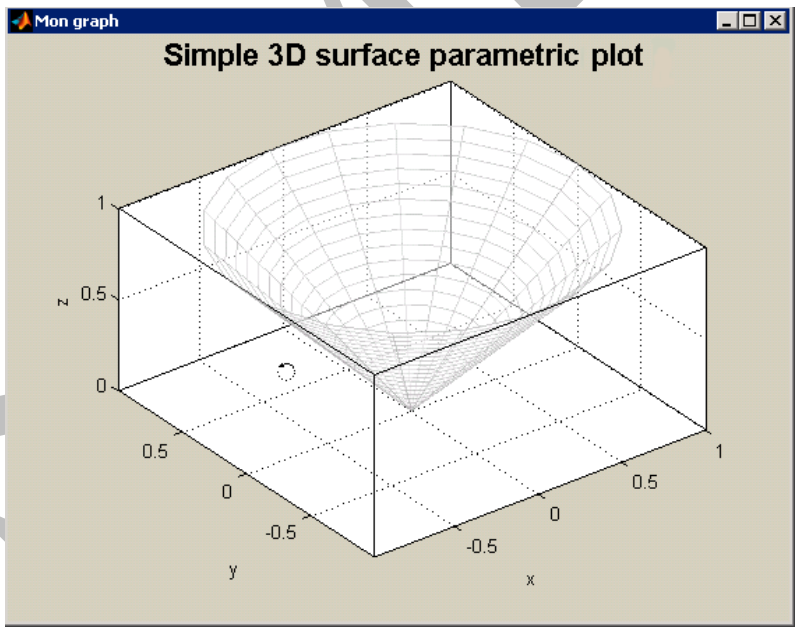


Now have a look of the logical continuity that is the cone but by adding transparency to the surface and the mesh:

SAM

```
1   close all;clear all;clc
2   figure
3   set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4       'Color',[0 0.8157 0.7483]);
5
6   r=linspace(0,1,20);
7   theta=linspace(0,2*pi,20);
8   %and r and theta becomes now a 20x20 matrix
9   [r,theta]=meshgrid(r,theta);
10
11  %we change from cylindric to cart. coordinates
12  x=r.*cos(theta);
13  y=r.*sin(theta);
14  z=r;
15
16  mhndl=mesh(x,y,z);
17  set(mhndl,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
18  xlabel('x');ylabel('y');zlabel('z');
19  title('Simple 3D surface parametrice plot','fontweight','bold','fontsize',16);
20  axis equal;
21  box on;
22  rotate3d on;
```

The result will looks like following:



Continue now with a little bit more sophisticated version near of what I saw once in General Relativity (a path on a curve surface) using a mix of the functions **meshgrid( )**, **mesh( )** and **line( )**:

```
close all;clear all;clc
figure
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);

r=linspace(0,1,20);
theta=linspace(0,2*pi,20);
[r,theta]=meshgrid(r,theta);

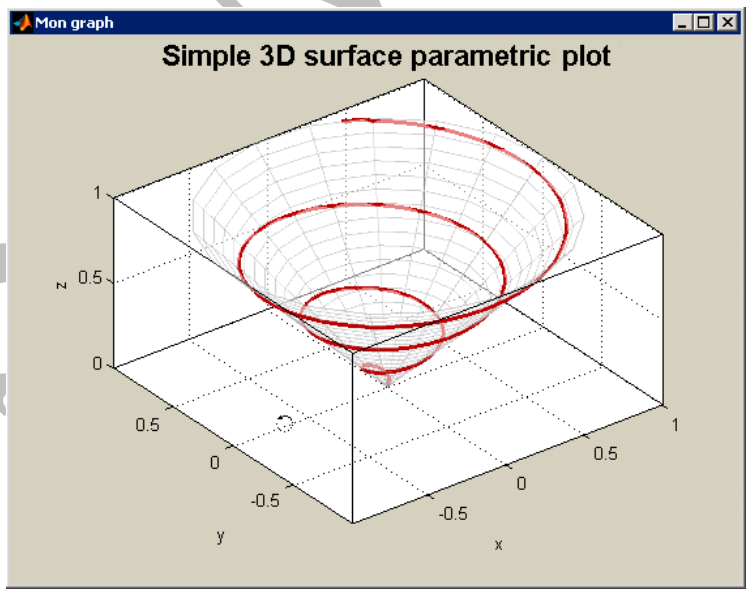
x=r.*cos(theta);y=r.*sin(theta);z=r;
mhndl=mesh(x,y,z);
set(mhndl,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);

%now we add a line on the surface and this can be donc only if the
%surface and the line trigoncometric functin have the sample amplitude
%(=1 in this example)
r=linspace(0,1,200);
x=r.*cos(20*r);y=r.*sin(20*r);z=r;

surfLine=line(x,y,z);
set(surfLine,'Color',[.625,0,0],'LineWidth',2)

xlabel('x');ylabel('y');zlabel('z');
title('Simple 3D surface parametrice plot','fontweight','bold','fontsize',16);
axis equal;
box on;
rotate3d on;
```

The result will looks like:



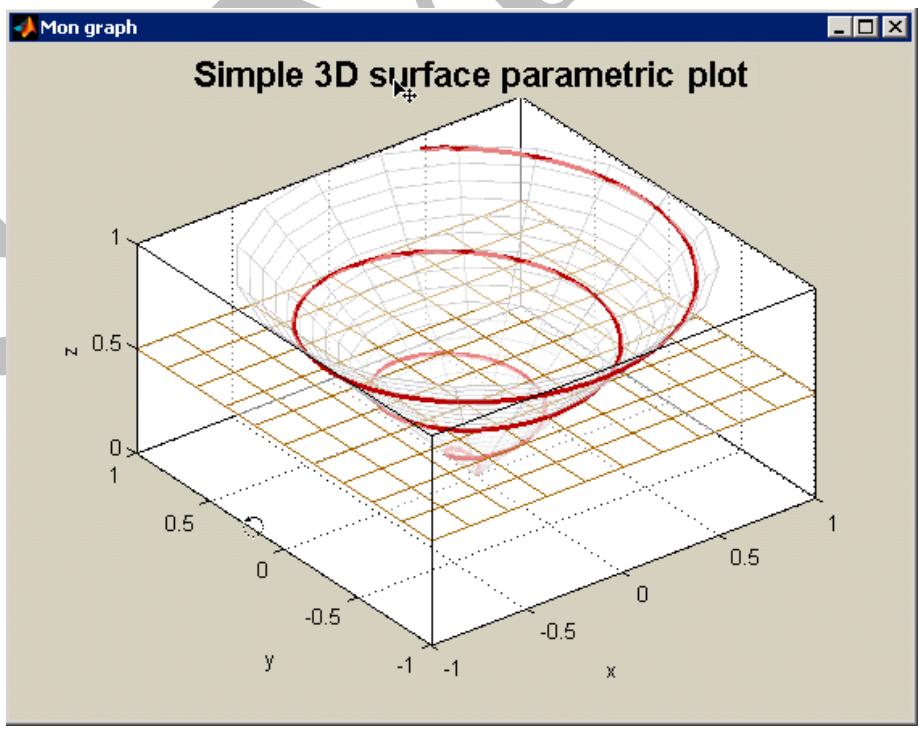
And now just for practice a composition of academics stuff (cone, line, plane):

EDITOR PUBLISH VIEW

```
close all;clear all;clc
figure
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);

%------------------------- cone
r=linspace(0,1,20);
theta=linspace(0,2*pi,20);
[r,theta]=meshgrid(r,theta);
x=r.*cos(theta);y=r.*sin(theta);z=r;
mhndl=mesh(x,y,z);
set(mhndl,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
%------------------------- line
r=linspace(0,1,200);
x=r.*cos(20*r);y=r.*sin(20*r);z=r;
surfLine=line(x,y,z);
set(surfLine,'Color',[.625,0,0],'LineWidth',2)
%------------------------- now we add a square plane at middle height
hold on
[x,y]=meshgrid(-1:0.2:1);
z=0.5*ones(size(x));
plane=mesh(x,y,z);
set(plane,'EdgeColor',[0.6,0.4,0],'FaceAlpha',0.5)

xlabel('x');ylabel('y');zlabel('z');
title('Simple 3D surface parametrice plot','fontweight','bold','fontsize',16);
axis equal;box on;rotate3d on;
```



This is also a very academic example to practice with a little bit more complicate surface (a tore):

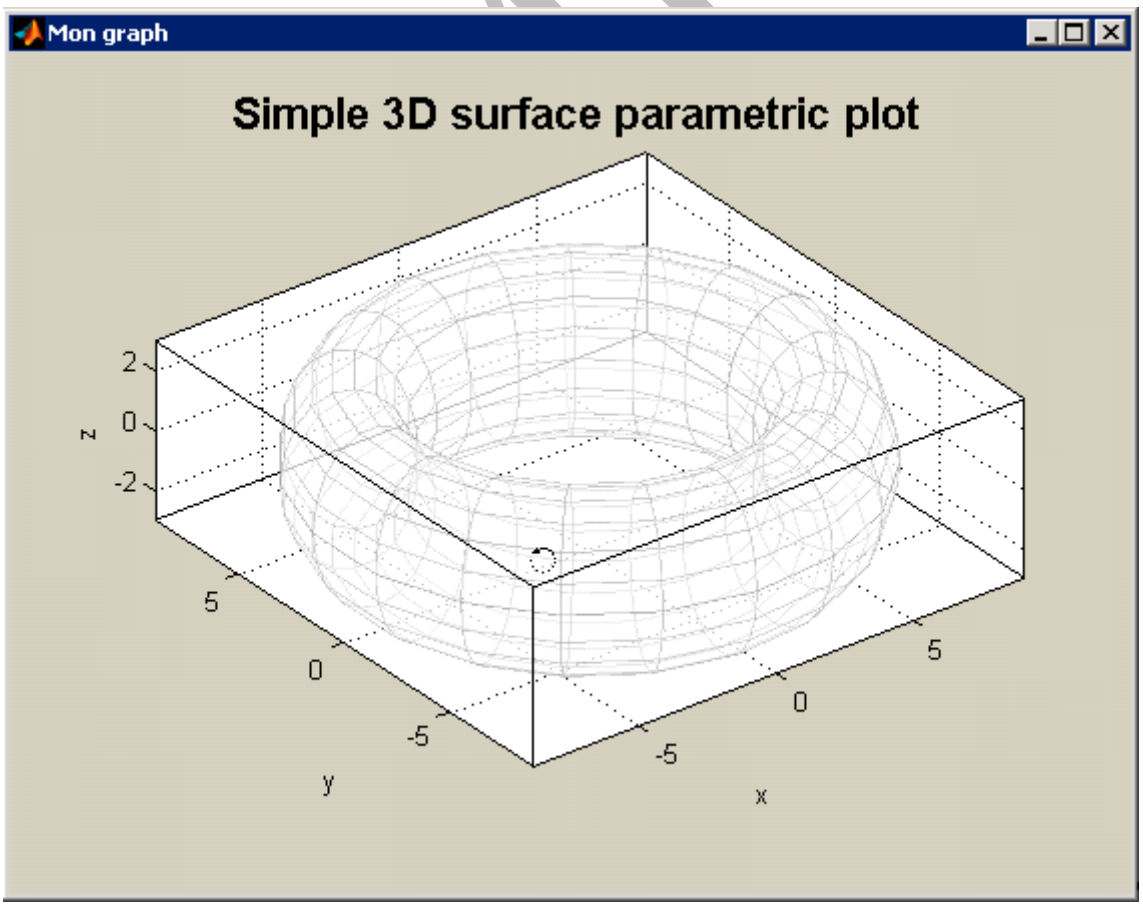
EDITOR PUBLISH VIEW

```
close all;clear all;clc
figure
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);

a=7;b=2;c=3;
u=linspace(0,2*pi,20);
v=linspace(0,2*pi,20);
[u,v]=meshgrid(u,v);
x=(a+b*cos(u)).*cos(v);
y=(a+b*cos(u)).*sin(v);
z=c*sin(u);

tore=mesh(x,y,z);
set(tore,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
axis equal

xlabel('x');ylabel('y');zlabel('z');
title('Simple 3D surface parametrice plot','fontweight','bold','fontsize',16);
axis equal;box on;rotate3d on;
```



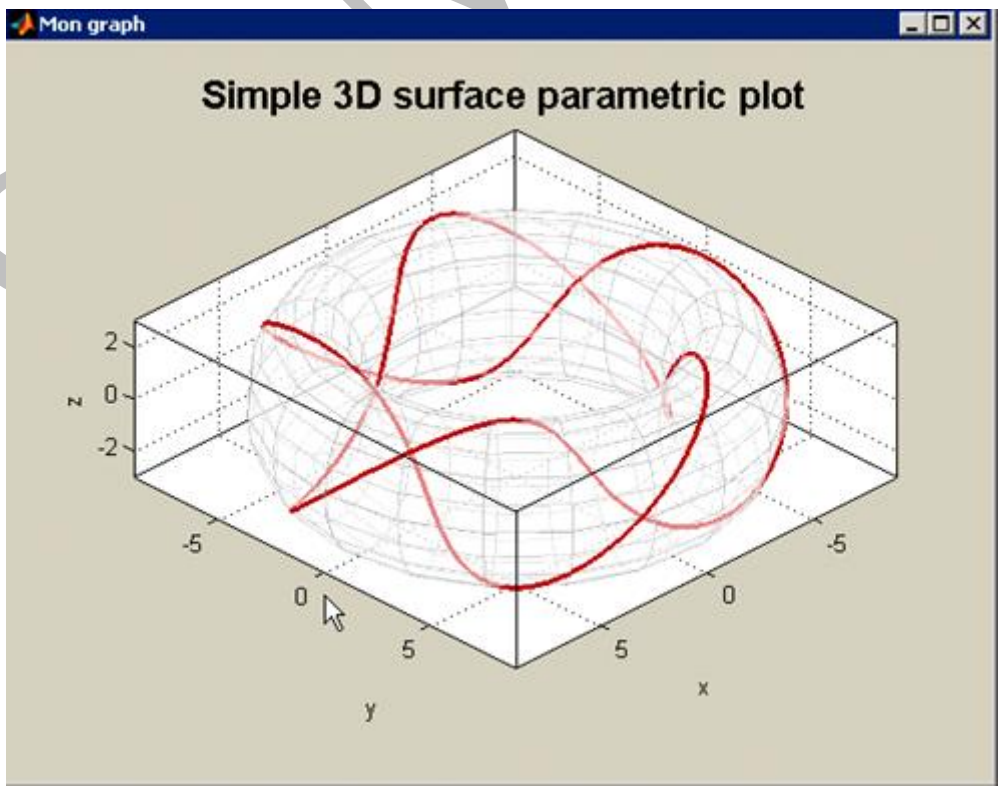
And always for practice purposes:

```
close all;clear all;clc
figure
set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
    'Color',[0.8314 0.8157 0.7483]);

%--------------- the tore
a=7;b=2;c=3;
u=linspace(0,2*pi,20);
v=linspace(0,2*pi,20);
[u,v]=meshgrid(u,v);
x=(a+b*cos(u)).*cos(v);
y=(a+b*cos(u)).*sin(v);
z=c*sin(u);
tore=mesh(x,y,z);
set(tore,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);

%--------------- a line on the tore
t=linspace(0,2*pi,200);
x=(a+b*cos(5*t)).*cos(2*t);
y=(a+b*cos(5*t)).*sin(2*t);
z=c*sin(5*t);
surfLine=line(x,y,z);
set(surfLine,'Color',[.625,0,0],'LineWidth',2)

xlabel('x');ylabel('y');zlabel('z');
title('Simple 3D surface parametrice plot','fontweight','bold','fontsize',16);
axis equal;box on;rotate3d on;view(135,30);
```



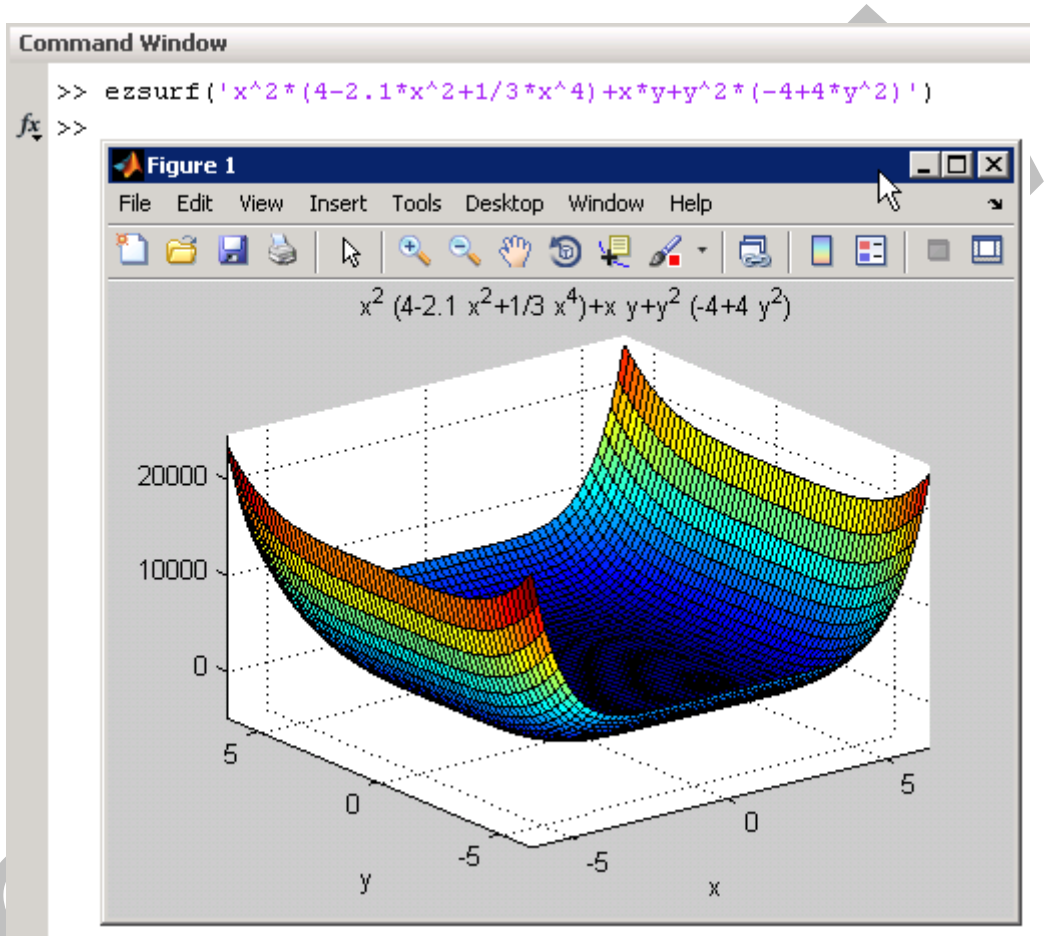
3D surface plots

I think it is not necessary to give examples of fields in which 3D surface charts are useful....

Then we start directly.

ezsurf

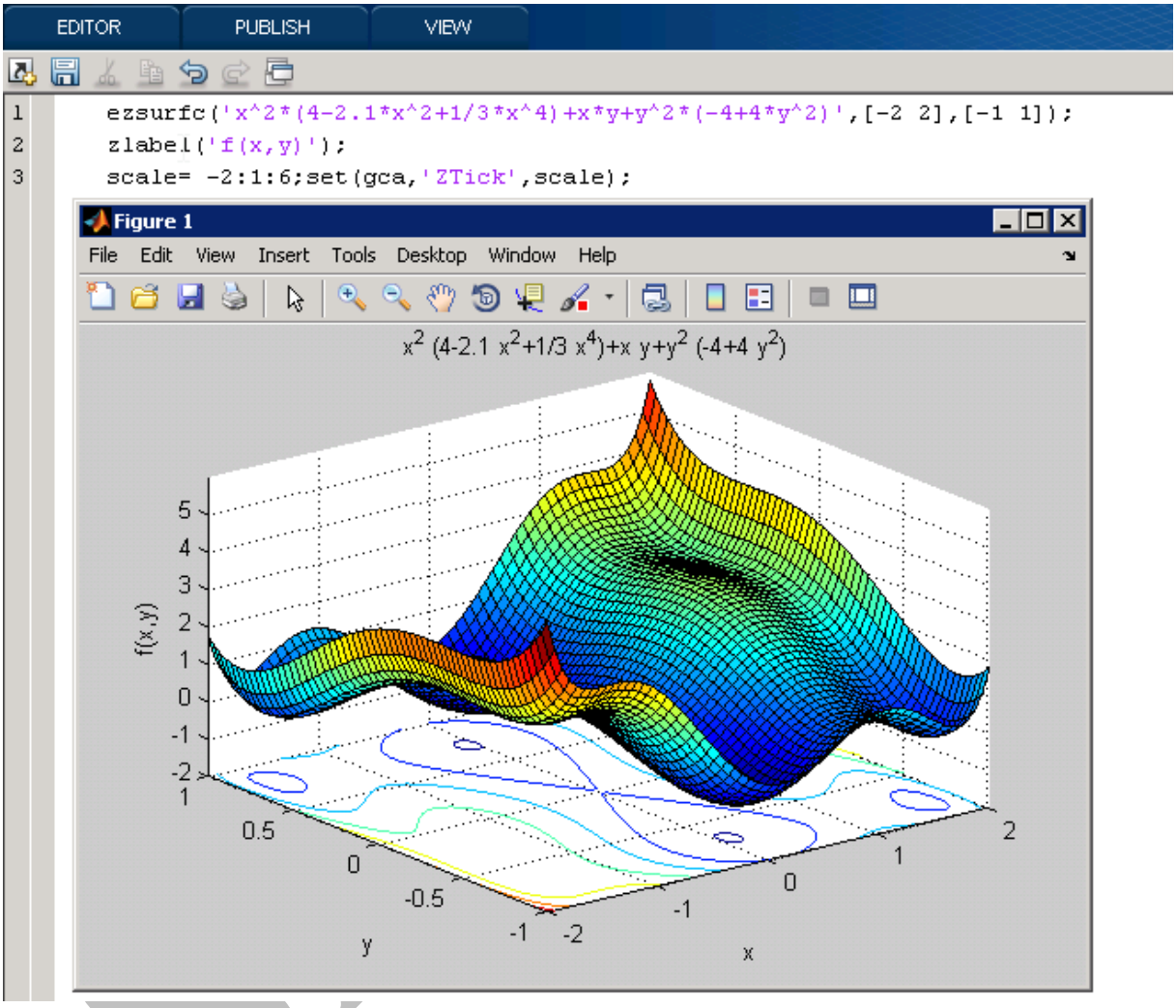
If you want to plot an analytical surface plot expression (typical in academics classes) you can just simply use the function **ezsurf( )**:



ezsurfc

The command **ezsurfc( )** also plot an analytical 3D surface plot expression but also add isoclines projected on the *XY* plane.

Let us see that with a few interesting customizations of the chart at the same time to learn also something more than just a simple function:



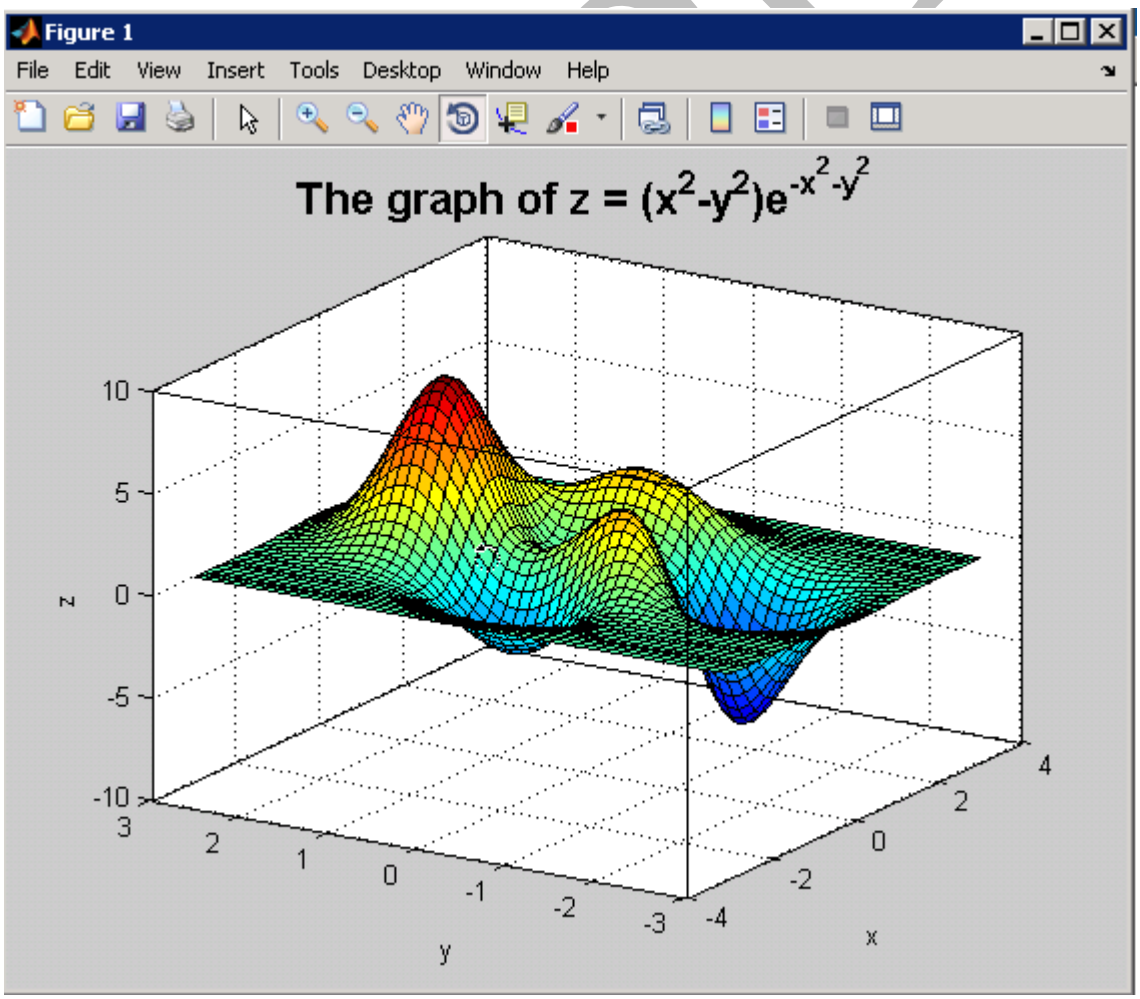
surf

The **surf( )** function will be useful when you have not an analytical expression but a grid of measures points (don't forget that x, y, z must be matrices!) and at the same time let us have a look to the command **view( )**:

EDITOR PUBLISH VIEW

```
1     [x, y ,z]=peaks;
2     surf(x,y,z);
3     title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4         'bold','fontsize',16);
5     box on;
6     xlabel('x');ylabel('y');zlabel('z');
7     rotate3d on;view(-58,24);
```

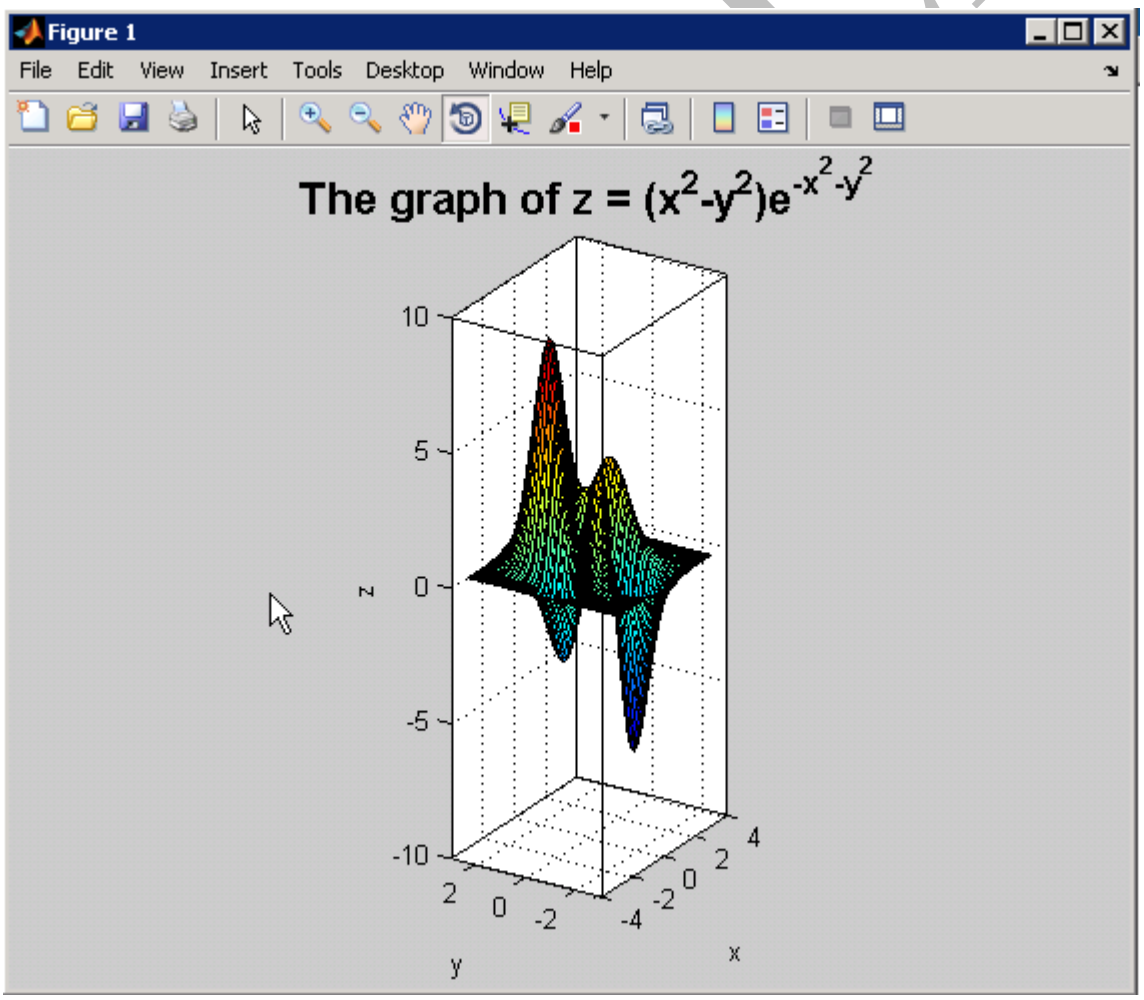
The result is the same as **ezsurf( )** excepted that the function is not analytically expressed:



As you can see on the previous chart axes are also not proportional. We can use the same technic as for 2D chart to respect them:

```
1     [x, y ,z]=peaks;
2     surf(x,y,z);
3     title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4         'bold','fontsize',16);
5     box on;
6     xlabel('x');ylabel('y');zlabel('z');
7     rotate3d on;view(-58,24);
8     axis equal;
9     axis([-4 4 -3 3 -10 10]);
```

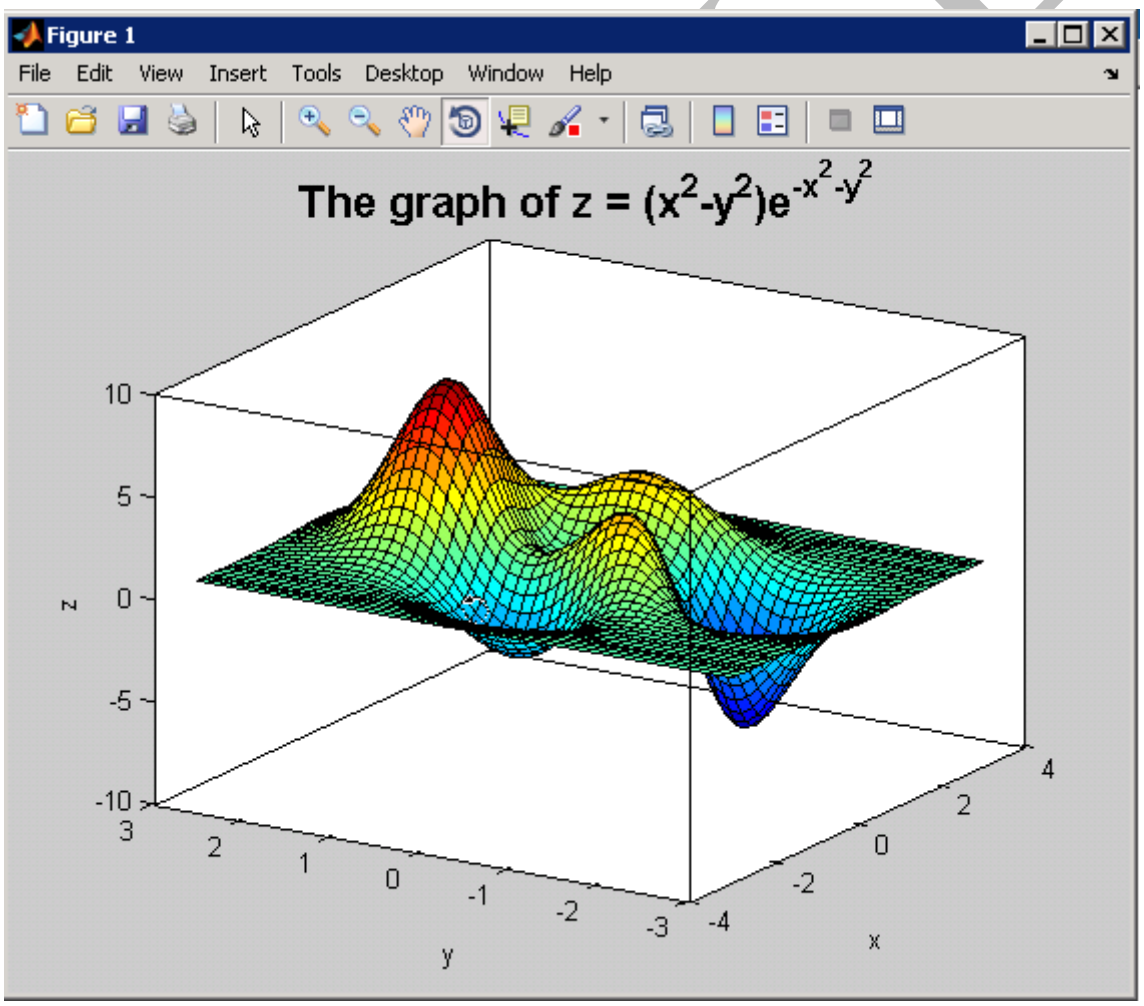
The result will be less pretty but more respective to real proportions:



Now see the result without grid just by curiosity:

```
1     [x, y ,z]=peaks;
2     surf(x,y,z);
3     title('The graph of z = (x^2-y^2)e^{-x^2-y^2}','fontweight',...
4         'bold','fontsize',16);
5     box on;
6     xlabel('x');ylabel('y');zlabel('z');
7     rotate3d on;view(-58,24);
8     grid off;
```

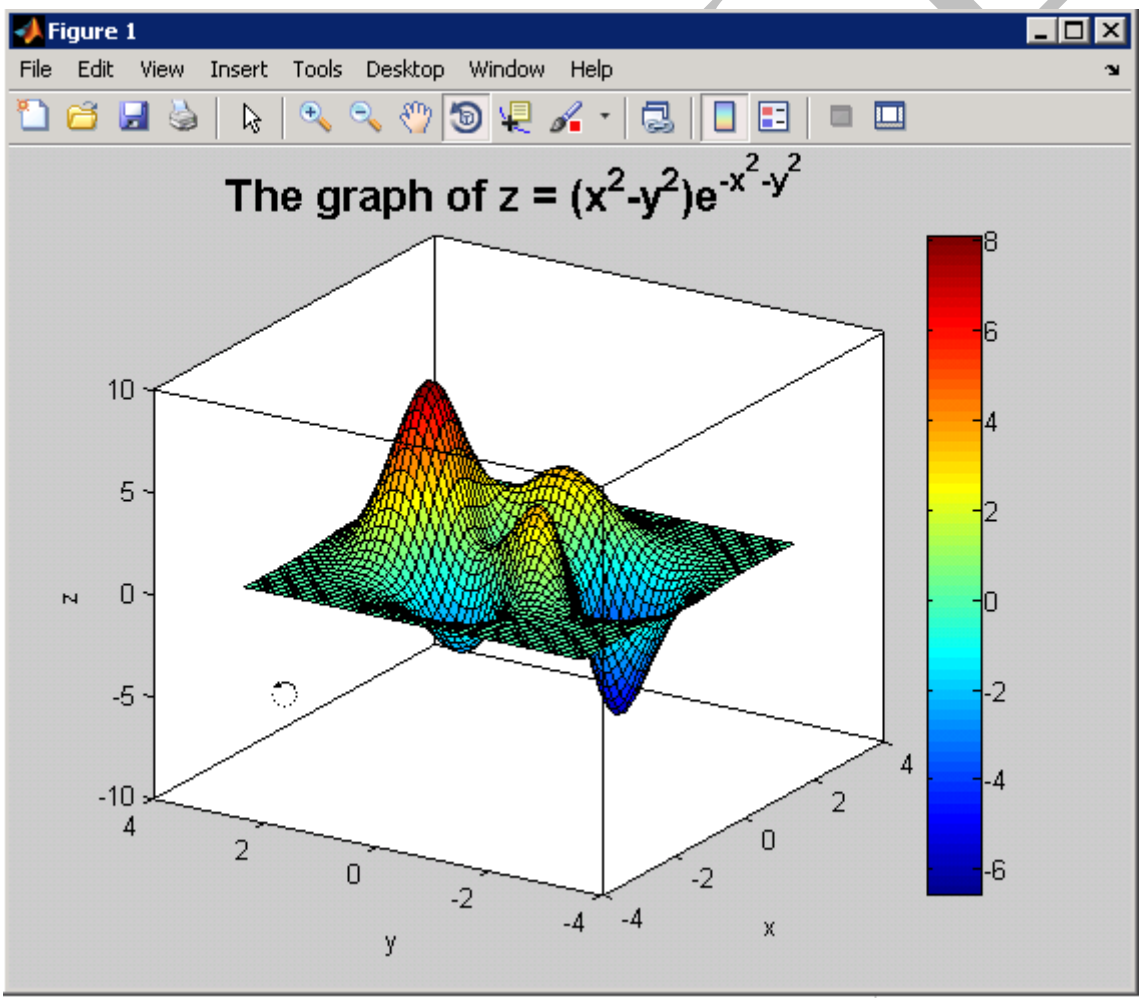
The result will be:



And how does it look like with a **colorbar**?:

```
1    [x, y ,z]=peaks;
2    surf(x,y,z);
3    title('The graph of z = (x^2-y^2)e^{-x^2-y^2}','fontweight',...
4        'bold','fontsize',16);
5    box on;
6    xlabel('x');ylabel('y');zlabel('z');
7    rotate3d on;view(-58,24);
8    grid off;
9    colorbar;
```

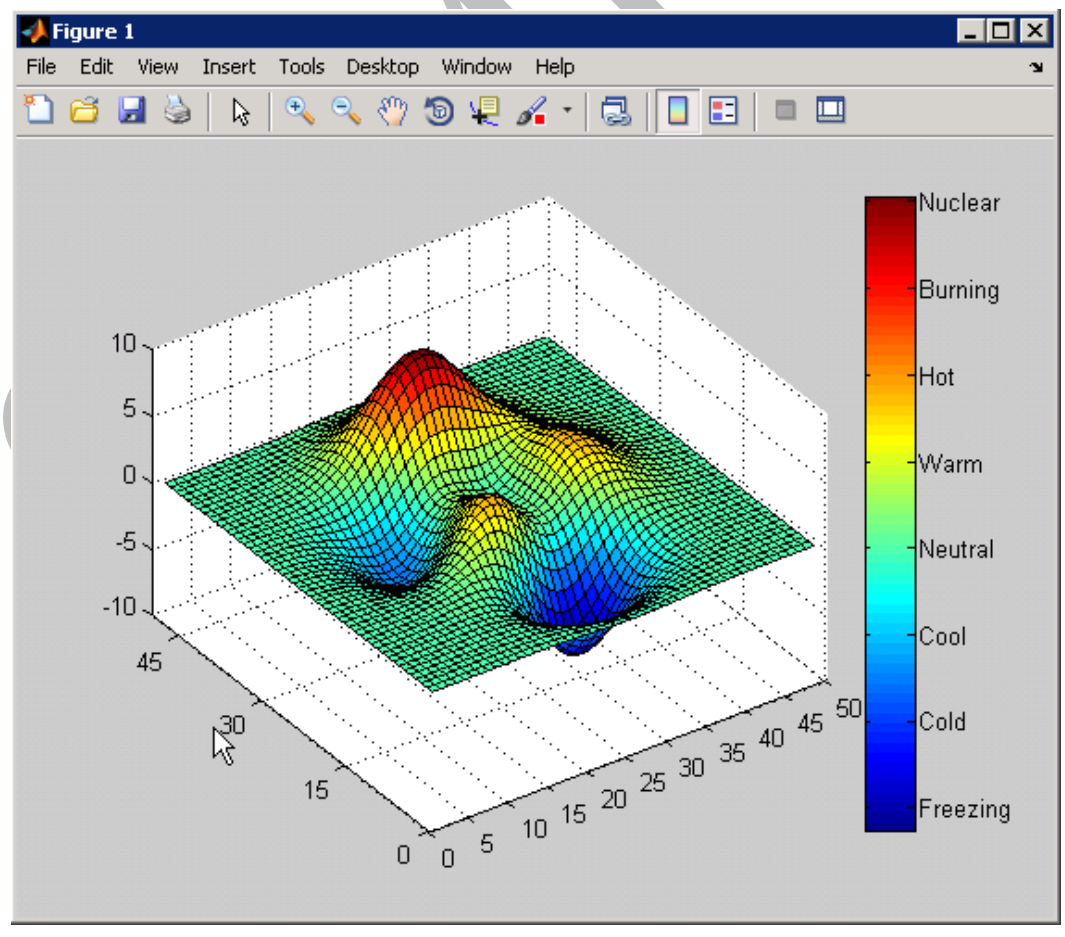
Not to bad



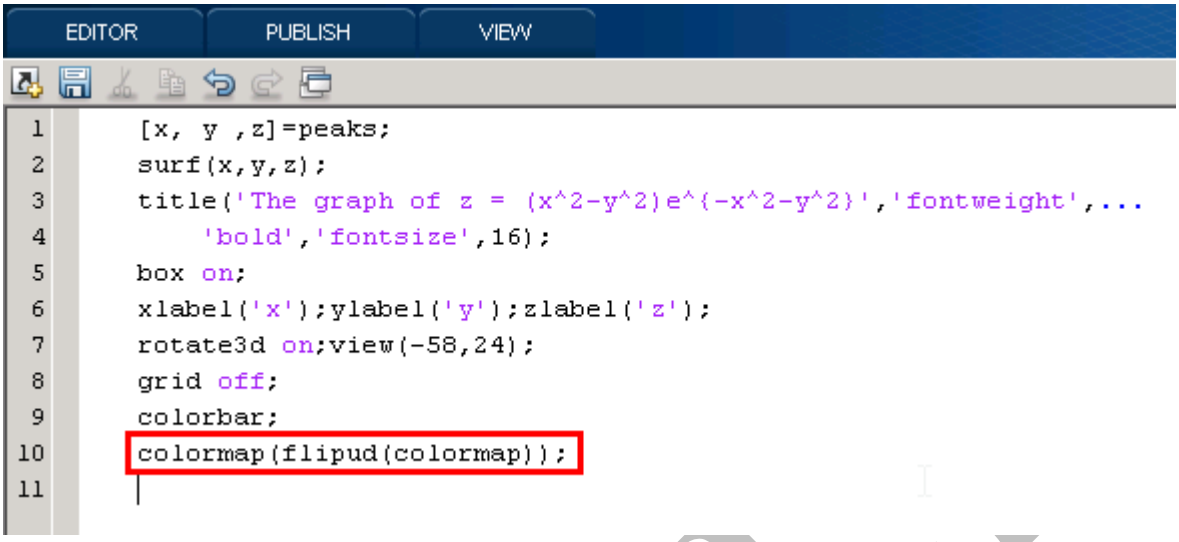
In reality (in practice...) we need sometimes words instead of value next to colorbars. See the trick:

```
1   clear all;close all;clc
2
3   surface(peaks)
4   colorbar('YTickLabel',...           % set labels to the colorbar
5       {'Freezing','Cold','Cool','Neutral',...
6        'Warm','Hot','Burning','Nuclear'})
7   view(-35,45)
8
9   number_panels = 0:5:50;
10  number_batteries = 0:15:50;
11
12  set(gca,'XTick',number_panels)      % set Xticks
13  set(gca,'YTick',number_batteries )  % set Yticks
14  grid on
```

This will give the expected result:



And if we want to invert the colors of the colorbar? We can use the function **flipud( )** for this purpose:

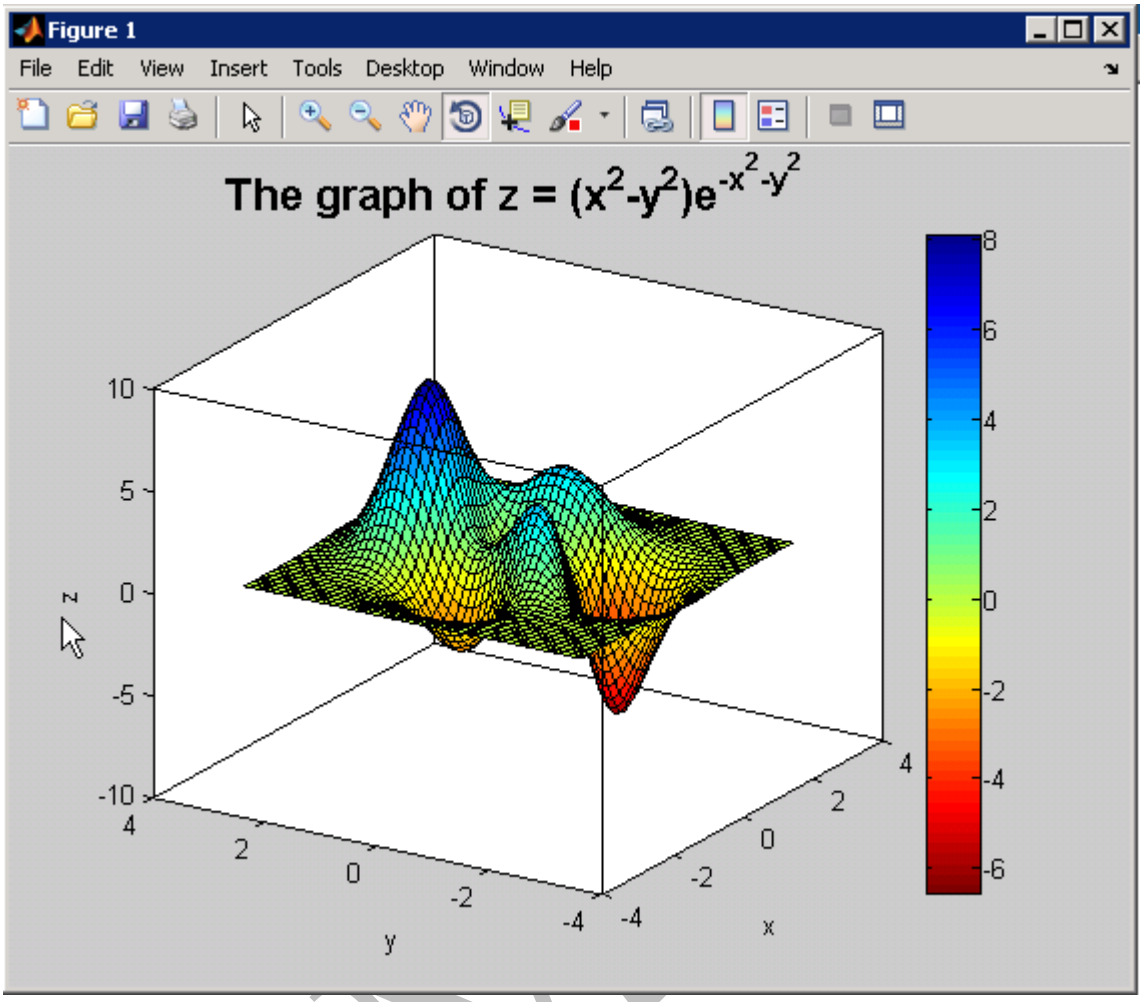


You can change the colors style of the colorbar by writing for example **colorbar('hot')**. Different styles exist:



The result will be as expected:

SAMPLE

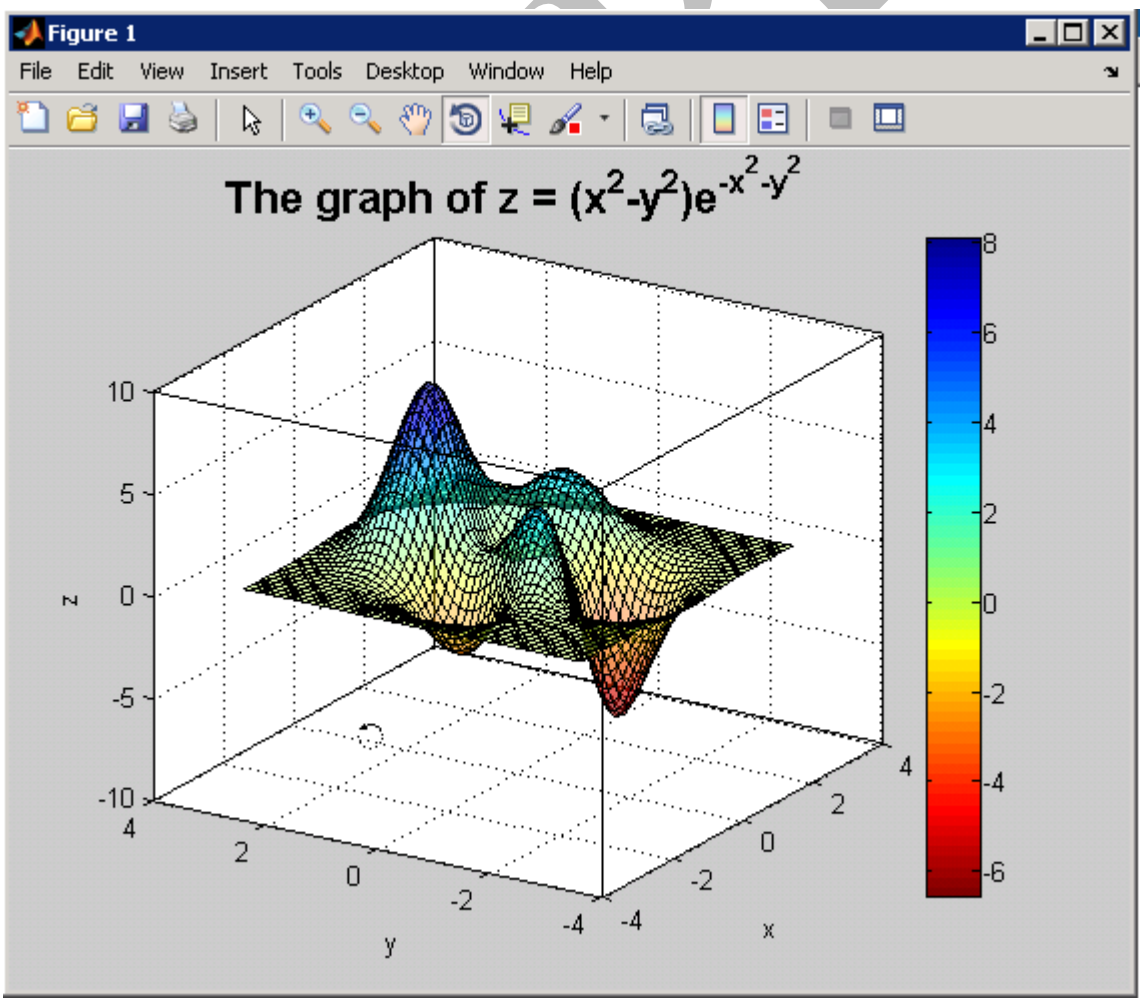

Figure 1
File Edit View Insert Tools Desktop Window Help
The graph of z = (x2-y2)e-x2-y2
10
5
0
-5
-10
z
4
2
0
-2
-4
y
-4
-2
0
2
4
x
8
6
4
2
0
-2
-4
-6

SAMPLE

And what about transparency if we play with the function **alpha( )**:

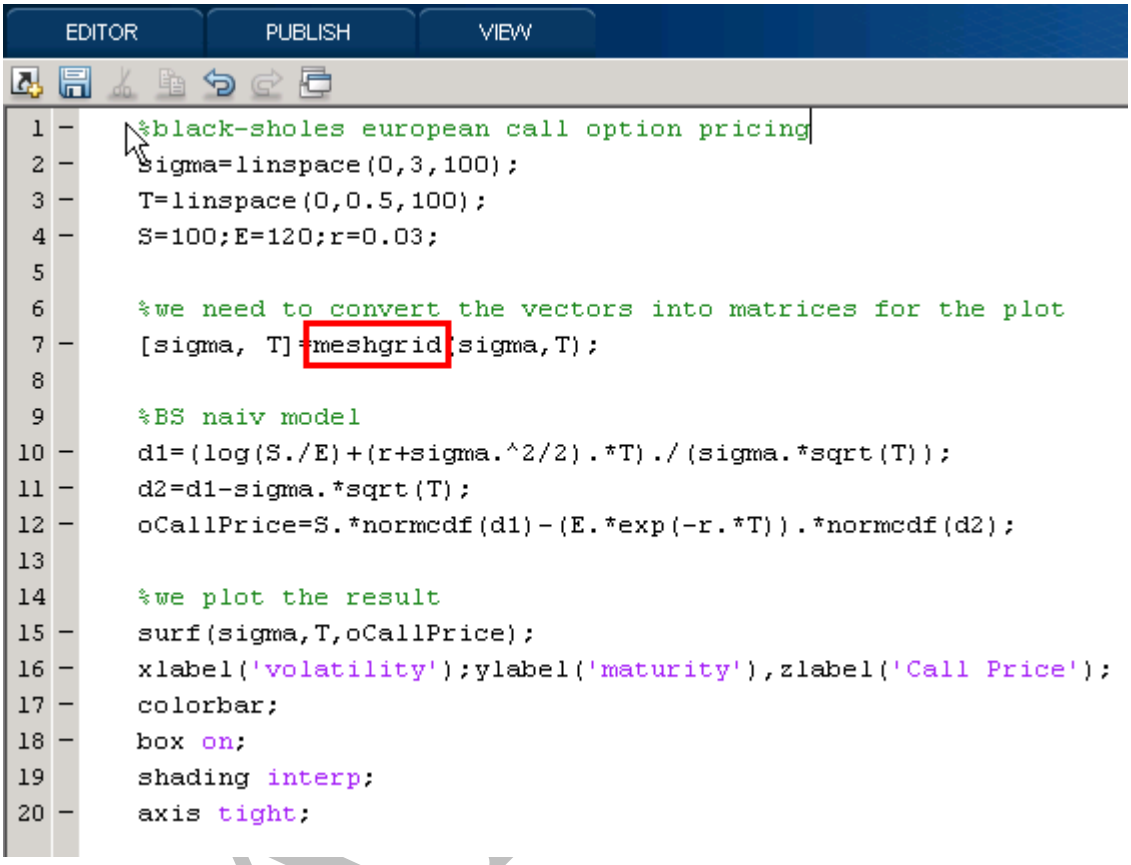
```
1 -    [x, y ,z]=peaks;
2 -    surf(x,y,z);
3      alpha(.4);
4 -    title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
5          'bold','fontsize',16);
6 -    box on;
7 -    xlabel('x');ylabel('y');zlabel('z');
8 -    rotate3d on;view(-58,24);
9 -    grid on;
10 -   colorbar;
11 -   colormap(flipud(colormap));
12     %to see trough the grid
13 -   hidden off;
```

We can see trough!:

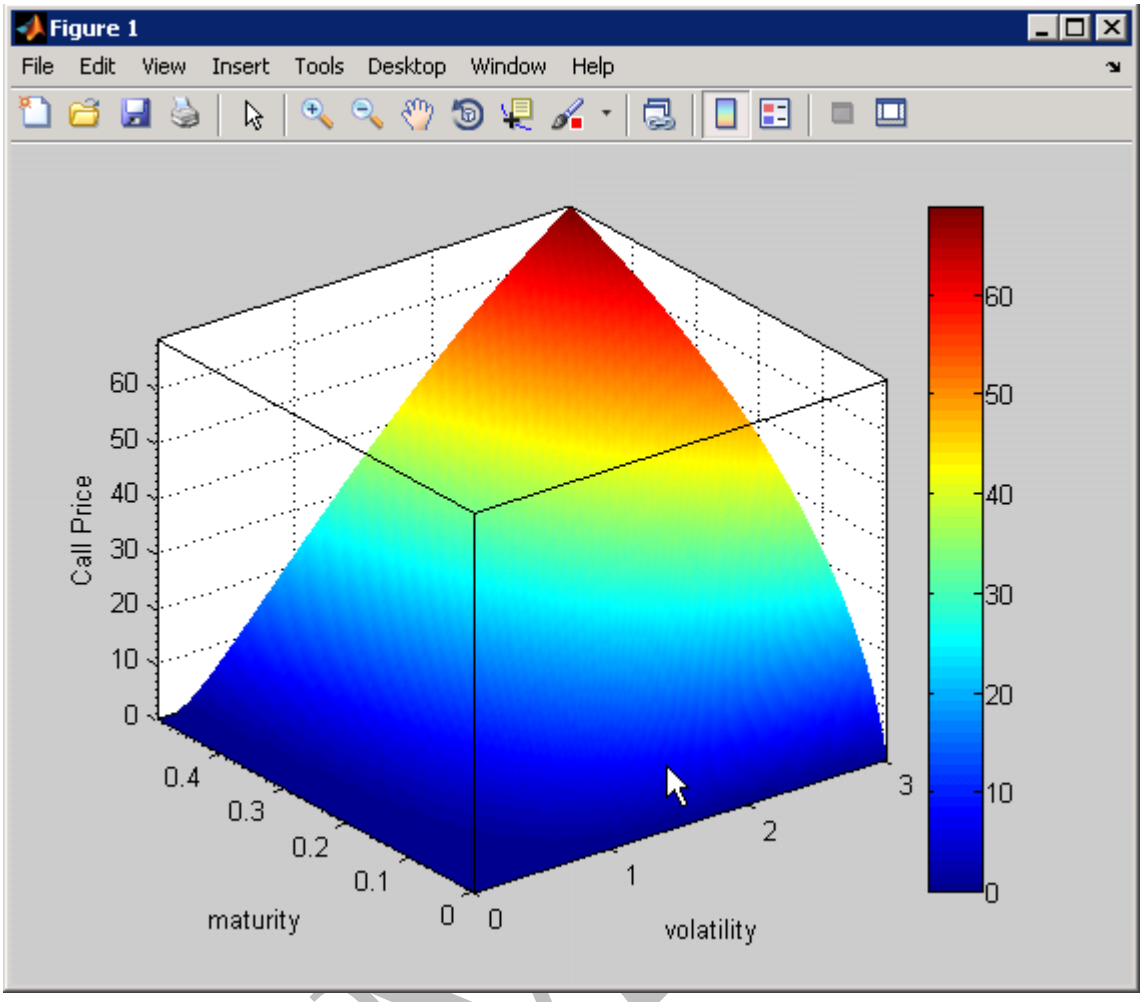


surf with an explicit or implicit function!

If you have an old version of MATLAB you won't be able to use **ezsurf( )**. Then here is method combining **meshgrid( )** and **surf( )** when you have a known function or only measures points to plot a surface and at the same time see how the command **shading interp** smooth the colors (for people working in laboratories and that have two vectors of parameters and one vector of measure you will have to wait a little bit):



SA



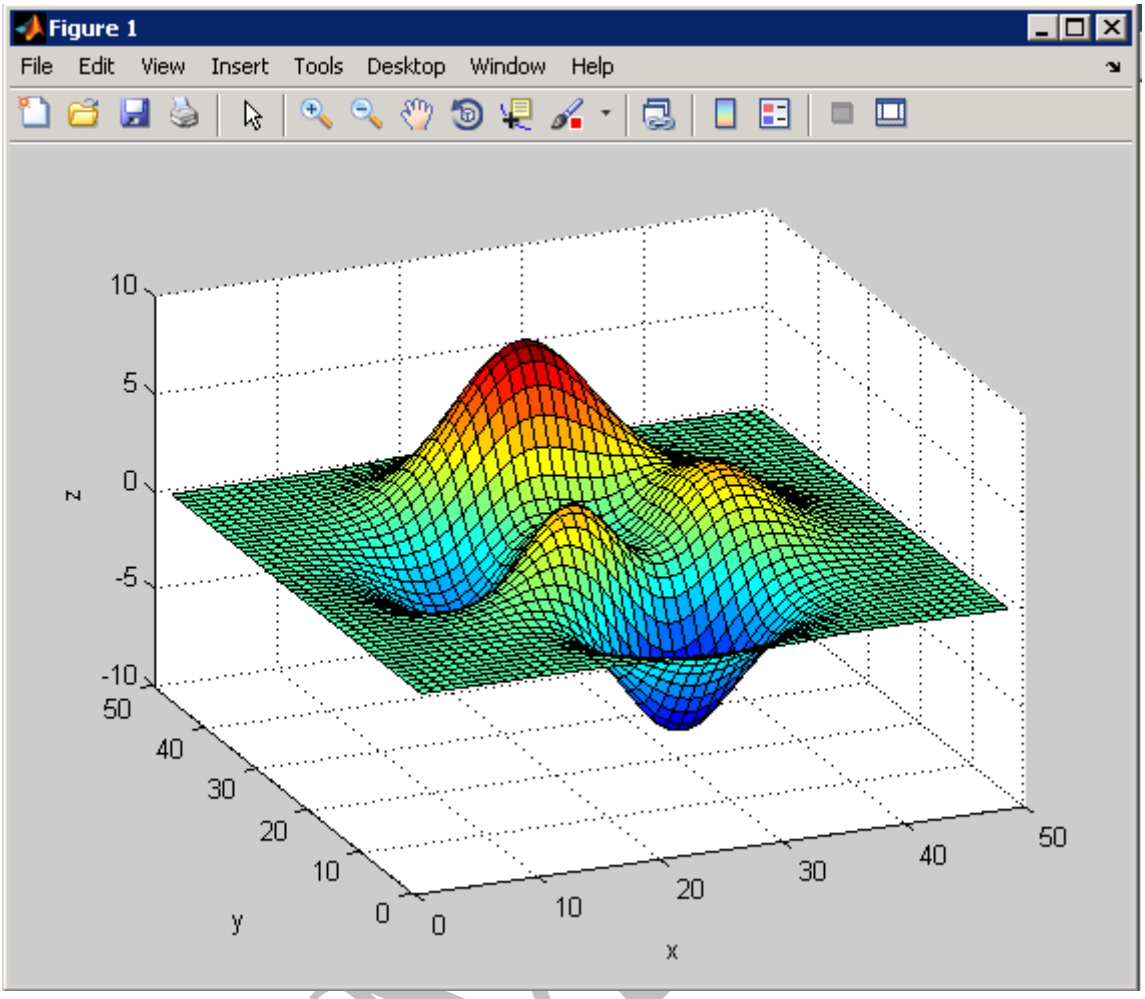
Animated surf

For users that what want to animate a rotating chart for their MS PowerPoint slides or their website (we will see later how to export the result as a *.gif) you can play with a loop and use the function **view( )** and the command **drawnow**:

EDITOR PUBLISH VIEW

```
clear all;close all;clc
Z=peaks;surf(Z);
% Keep axes constant
axis([0 50 0 50 -10 10])
% Define title and labels for reference
xlabel('x'); ylabel('y'); zlabel('z')
% Modify azimuth (horizontal rotation) and update drawing
for az = -37.5 : .5 : 30
    view(az, 30)
    drawnow
end
```

For sure you won't be able to see the animation work in this PDF:

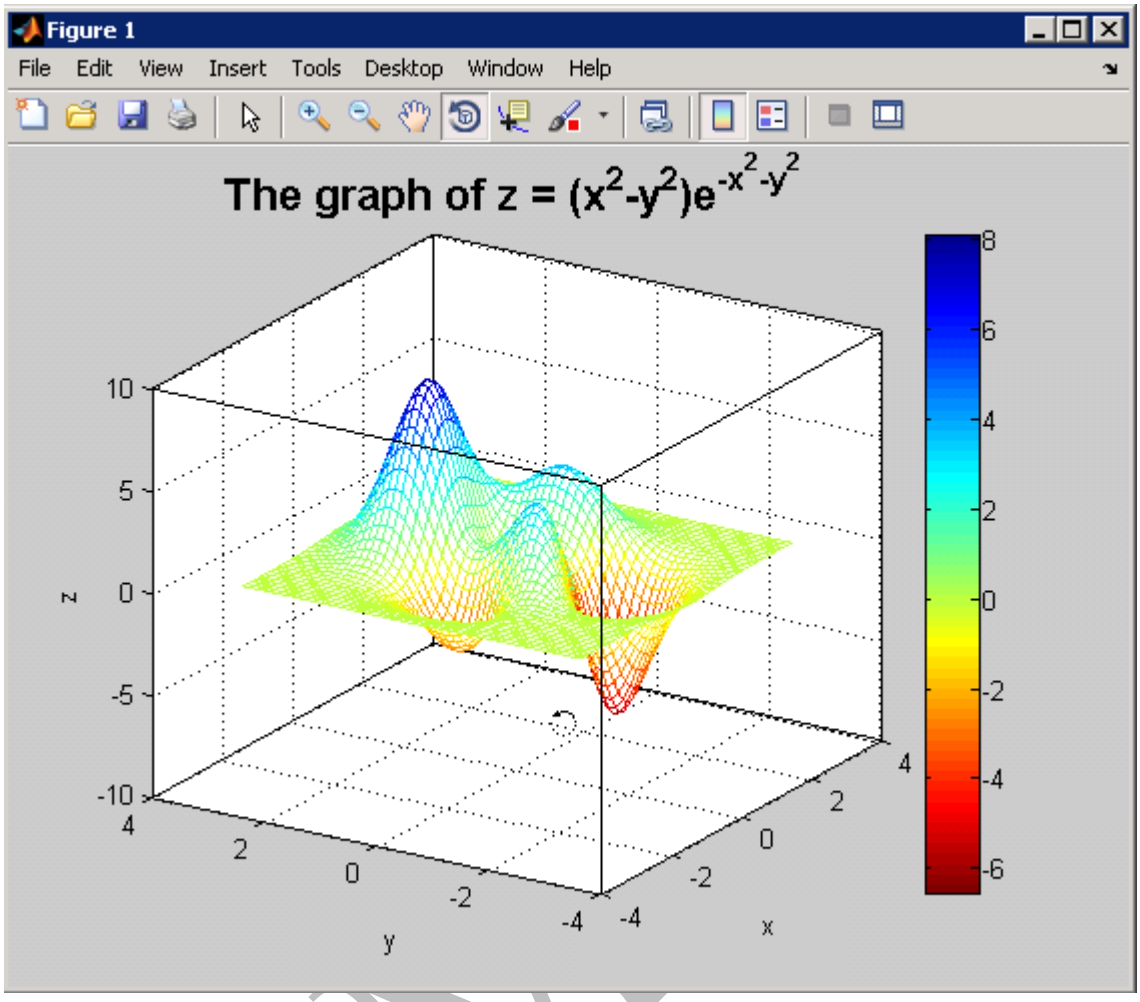


mesh

The **mesh( )** function is very important because it fits like for surf( ) not a surface but a surface grid:

```
EDITOR  PUBLISH  VIEW

1   [x, y ,z]=peaks;
2   mesh(x,y,z);
3   title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4       'bold','fontsize',16);
5   box on;
6   xlabel('x');ylabel('y');zlabel('z');
7   rotate3d on;view(-58,24);
8   grid on;
9   colorbar;
10  colormap(flipud(colormap));
11
```

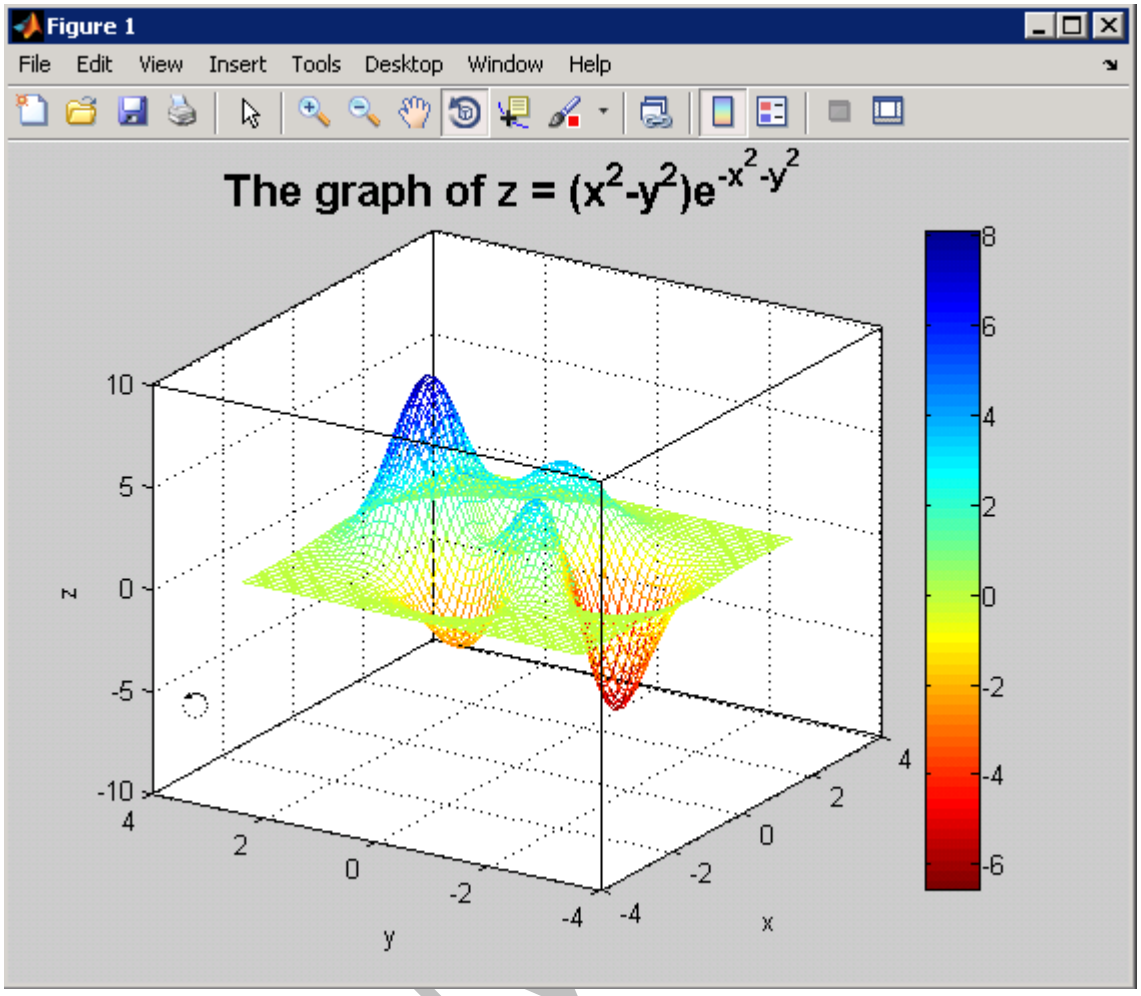


The grid is visible but we can't see trough. If we add **hidden off**:

EDITOR PUBLISH VIEW

```
1   [x, y ,z]=peaks;
2   mesh(x,y,z);
3   title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4       'bold','fontsize',16);
5   box on;
6   xlabel('x');ylabel('y');zlabel('z');
7   rotate3d on;view(-58,24);
8   grid on;
9   colorbar;
10  colormap(flipud(colormap));
11  %to see trough the grid
12  hidden off;
```

The result is as expected:

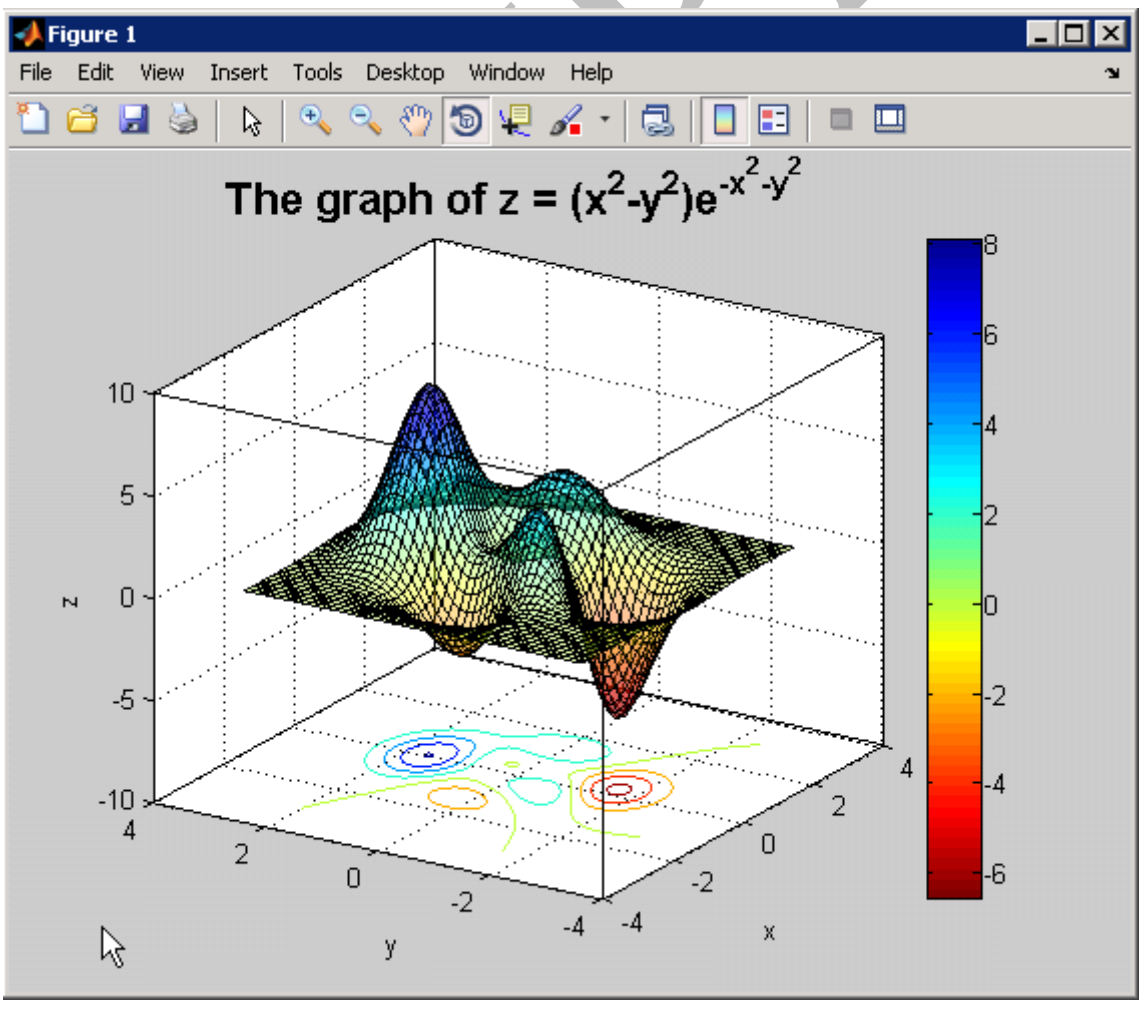


SAM

surfc

For the same reason as the function **surf( )** exists, you can use **surfc( )** to have isoclines on the bottom plane:

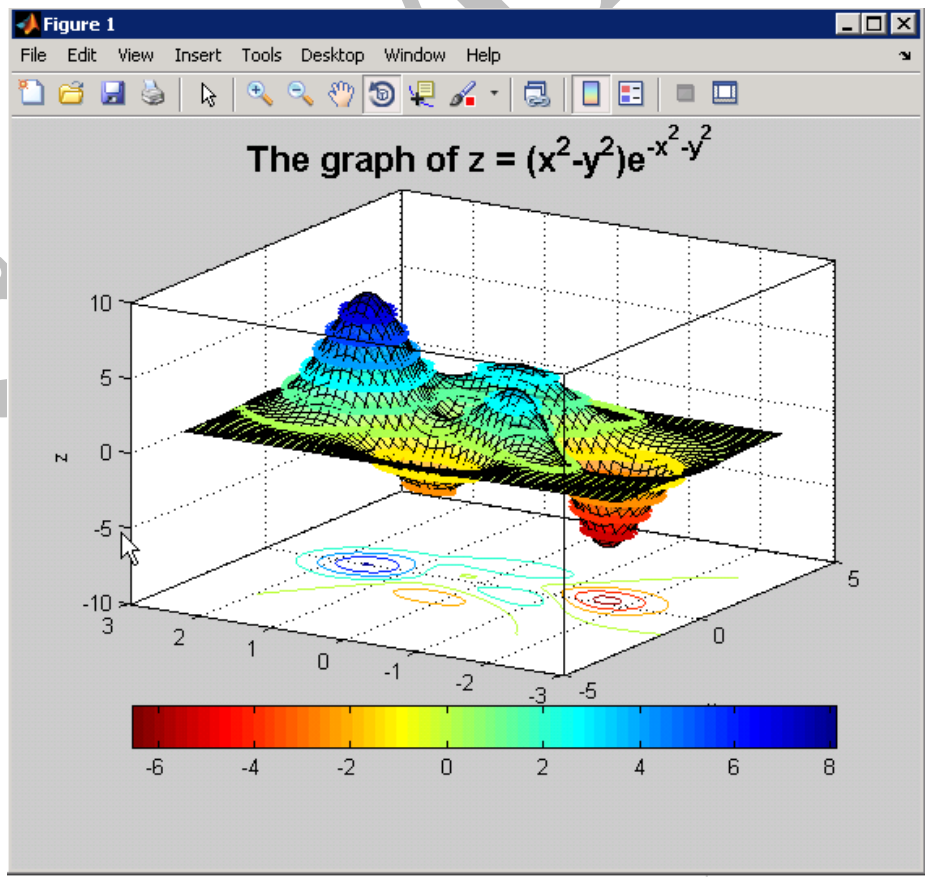
```
1 -   [x, y ,z]=peaks;
2 -   surfc(x,y,z);
3     alpha(.4);
4 -   title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
5         'bold','fontsize',16);
6 -   box on;
7 -   xlabel('x');ylabel('y');zlabel('z');
8 -   rotate3d on;view(-58,24);
9 -   grid on;
10 -  colorbar;
11 -  colormap(flipud(colormap));
12    %to see trough the grid
13 -  hidden off;
```



contour3 (3d contours)

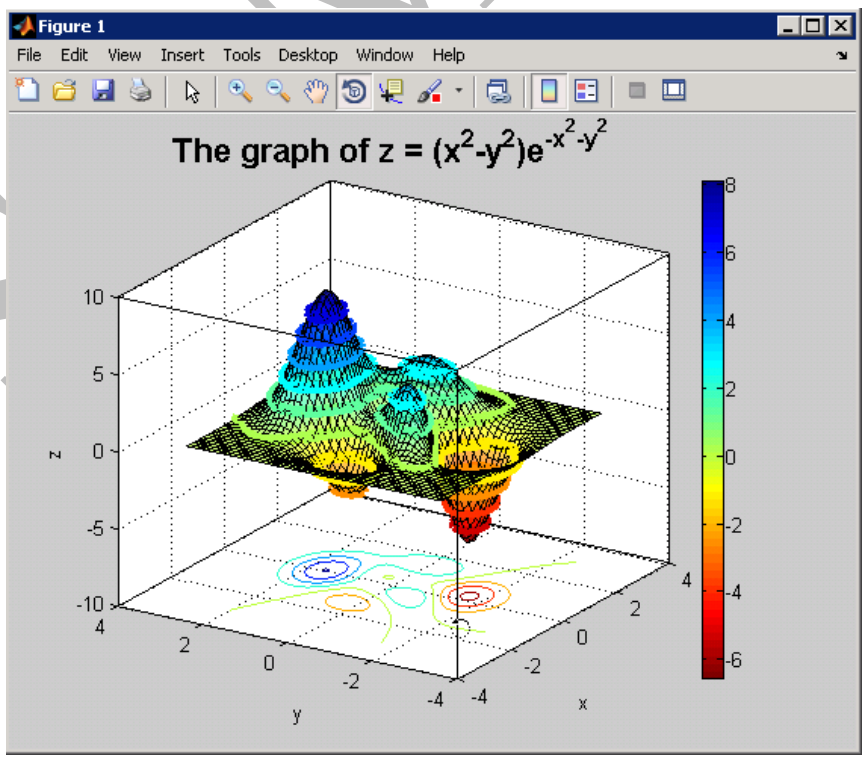
The function **contour3( )** is especially used by geophysicists and geometricians:

```
1  -  [x, y ,z]=peaks;
2  -  surfc(x,y,z);
3  -  hold on
4     %contours with 10 isoclines
5  -  [c,h]=contour3(x,y,z,10);
6  -  set(h,'linewidth',5);
7  -  alpha(0.7);
8  -  title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
9        'bold','fontsize',16);
10 -  box on;
11 -  xlabel('x');ylabel('y');zlabel('z');
12 -  rotate3d on;view(-58,24);
13 -  grid on;
14 -  colorbar('location','southoutside');
15 -  colormap(flipud(colormap));
16    %to see trough the grid
17 -  hidden off;
```



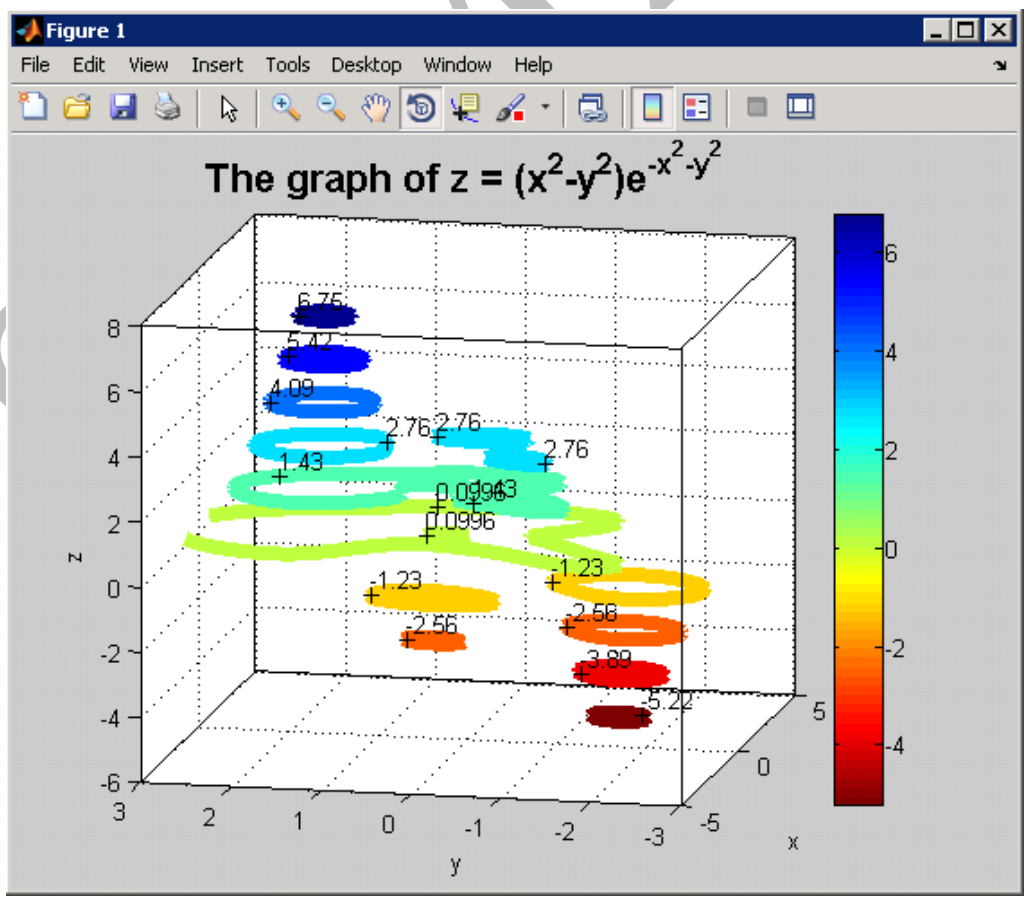
Now see the trick... to control the colorbar width (almost impossible to guess):

```
1  -    [x, y ,z]=peaks;
2  -    surfc(x,y,z);
3  -    hold on
4       %contours with 10 isoclines
5  -    [c,h]=contour3(x,y,z,10);
6  -    set(h,'linewidth',5);
7  -    alpha(0.7);
8  -    title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
9           'bold','fontsize',16);
10 -    box on;
11 -    xlabel('x');ylabel('y');zlabel('z');
12 -    rotate3d on;view(-58,24);
13 -    grid on;
14 -    c =colorbar;
15 -    axpos = get(gca,'Position');
16 -    cpos = get(c,'Position');
17 -    cpos(3) = 0.5*cpos(3);
18 -    set(c,'Position',cpos)
19 -    set(gca,'Position',axpos)
20 -    colormap(flipud(colormap));
21      %to see trough the grid
22 -    hidden off;
```

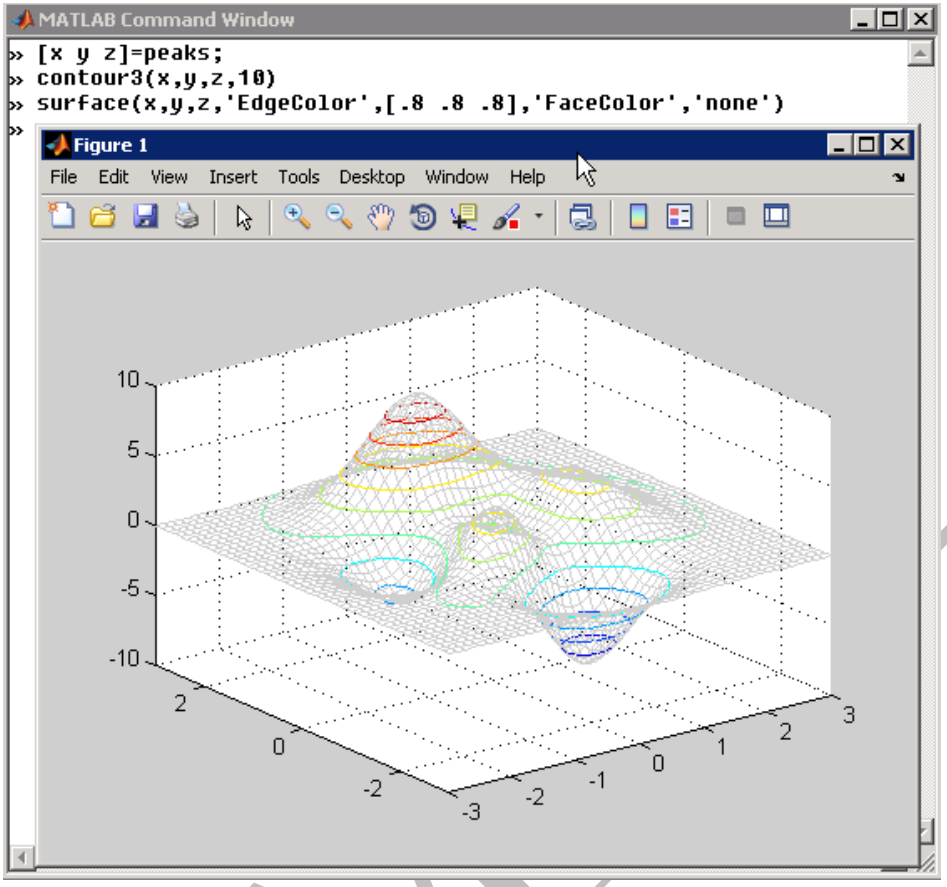


Most time geophysicists and geometricians don't what the surface but just the isoclines. Then it is very easy. You just remove the **surf( )** or **surfc( )** function of your script and to make the values of the contours on the chart you use the function **clabel( )**:

```
[x, y ,z]=peaks;
hold on
%contours avec 10 isoclines
[c,h]=contour3(x,y,z,10);
clabel(c)
set(h,'linewidth',5);
alpha(0.7);
title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
    'bold','fontsize',16);
box on;
xlabel('x');ylabel('y');zlabel('z');
rotate3d on;view(-78,14);
grid on;
colorbar;
colormap(flipud(colormap));
%to see trough the grid
hidden off;
```



Or another nice result:



surf with surf projection

Many geophysicists and physicists need to have a surface plot with at the same time a surface density plot projection. This option does not exists in MATLAB then we have to use a trick to get the expected result:

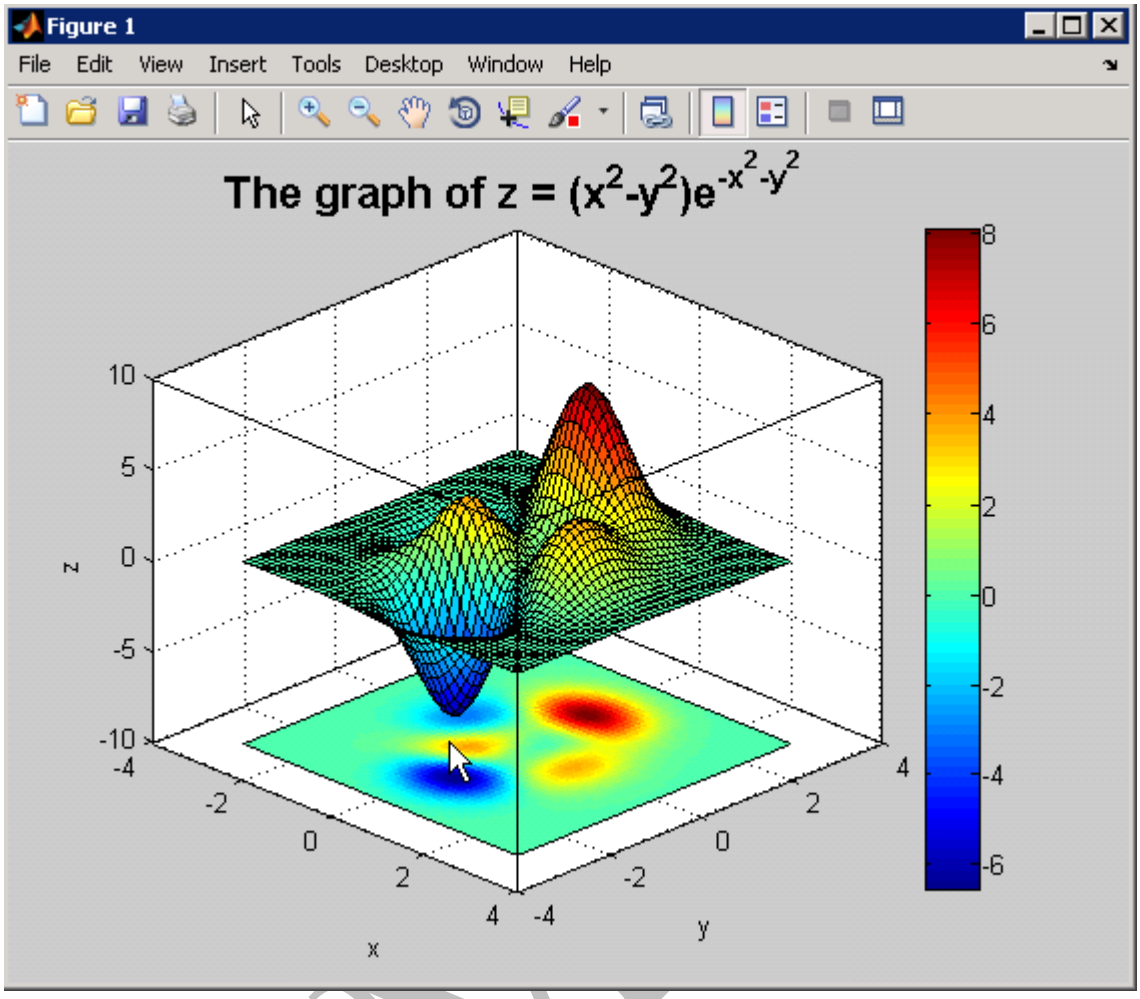
SAMPLE

EDITOR PUBLISH VIEW

```
1  -    [x,y,z] = peaks;
2       % get the corners of the domain in which the data occurs.
3  -    min_x = min(min(x));
4  -    min_y = min(min(y));
5  -    max_x = max(max(x));
6  -    max_y = max(max(y));
7         figure
8       % do a normal surface plot.
9  -    surf(x,y,z);
10 -    hold on;
11
12      % the image data you want to show as a plane.
13      % especially this command is almost very difficulit to guess
14 -    planeimg = z;
15      % desired z position of the image plane.
16 -    imgzposition = -10;
17
18      % plot the image plane using surf.
19 -    surf([min_x max_x],[min_y max_y],repmat(imgzposition, [2 2]),...
20          planeimg,'facecolor','texture')
21
22 -    colorbar;
23      title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
24      'bold','fontsize',16);
25 -    box on;grid on;
26 -    view(45,30);
27 -    xlabel('x');ylabel('y');zlabel('z');
```

The result is on the next page:

SA


Figure 1
File Edit View Insert Tools Desktop Window Help
The graph of z = (x2-y2)e-x2-y2
10
5
0
-5
-10
z
-4
-2
0
2
4
x
-4
-2
0
2
4
y
8
6
4
2
0
-2
-4
-6

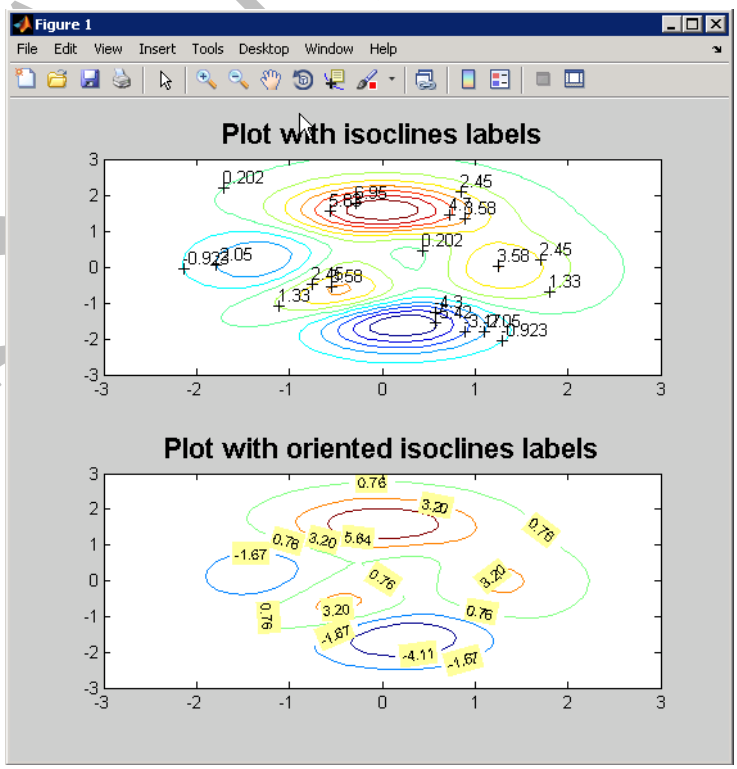
SAMPLE

contour (2d contours)

The function **contour( )** works in the same idea as the **contour3( )** that we was before, the difference is that this first function do the job in a projected *XY* plane:

```
1   subplot(2,1,1);
2   [X,Y,Z]=peaks;
3   %12 contour lines
4   C=contour(X,Y,Z,12);
5   clabel(C)
6   title('Plot with isoclines labels','fontweight','bold','fontsize',16)
7
8   subplot(2,1,2)
9   %5 oriented contour lines
10  [C,h]=contour(X,Y,Z,5);
11  texth=clabel(C,h,'fontsize', 8)
12  set(texth,'BackgroundColor',[1 1 .6])
13  %loop for decimal precision control of labels...
14  for i=1:size(texth)
15      textstr=get(texth(i),'String');
16      textnum=str2double(textstr);
17      textstrnew=sprintf('%0.2f', textnum);
18      set(texth(i),'String',textstrnew);
19  end
20  title('Plot with oriented isoclines labels','fontweight',...
21      'bold','fontsize',16)
```

The result looks like:

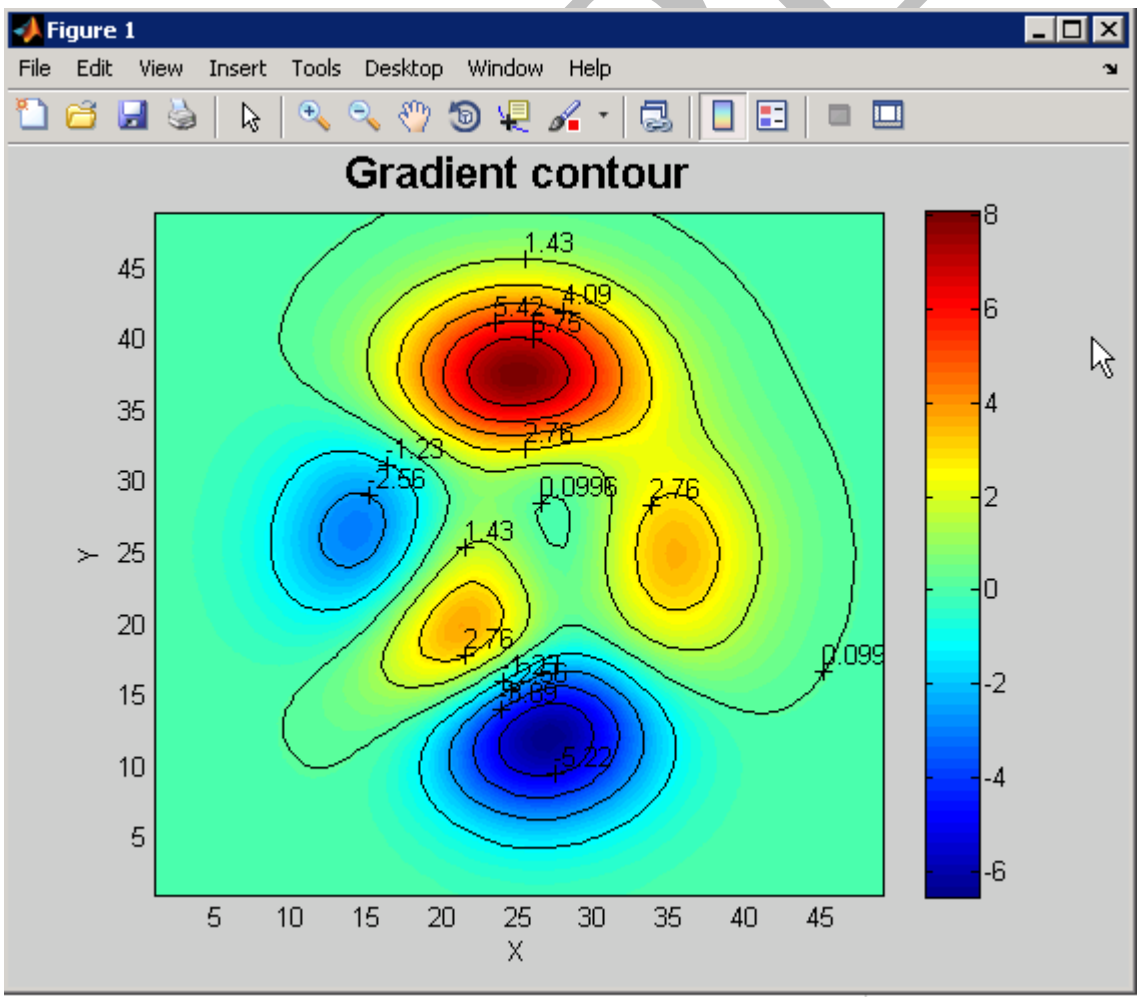


Note: It seems that we can't actually control easily the number of labels that appears automatically on a isocline (contour)

On a more pretty way using a colorbar and also a shading interpolation:

```
1   h=peaks;
2   pcolor(h);
3   colorbar;
4   grid on;
5   shading interp;
6   hold on;
7   C=contour(h,10,'k');
8   clabel(C);
9   title('Gradient contour','fontweight','bold','fontsize',16);
10  xlabel('X'),ylabel('Y')
```

This will give (**ToDo**: found how to activate the grid...):



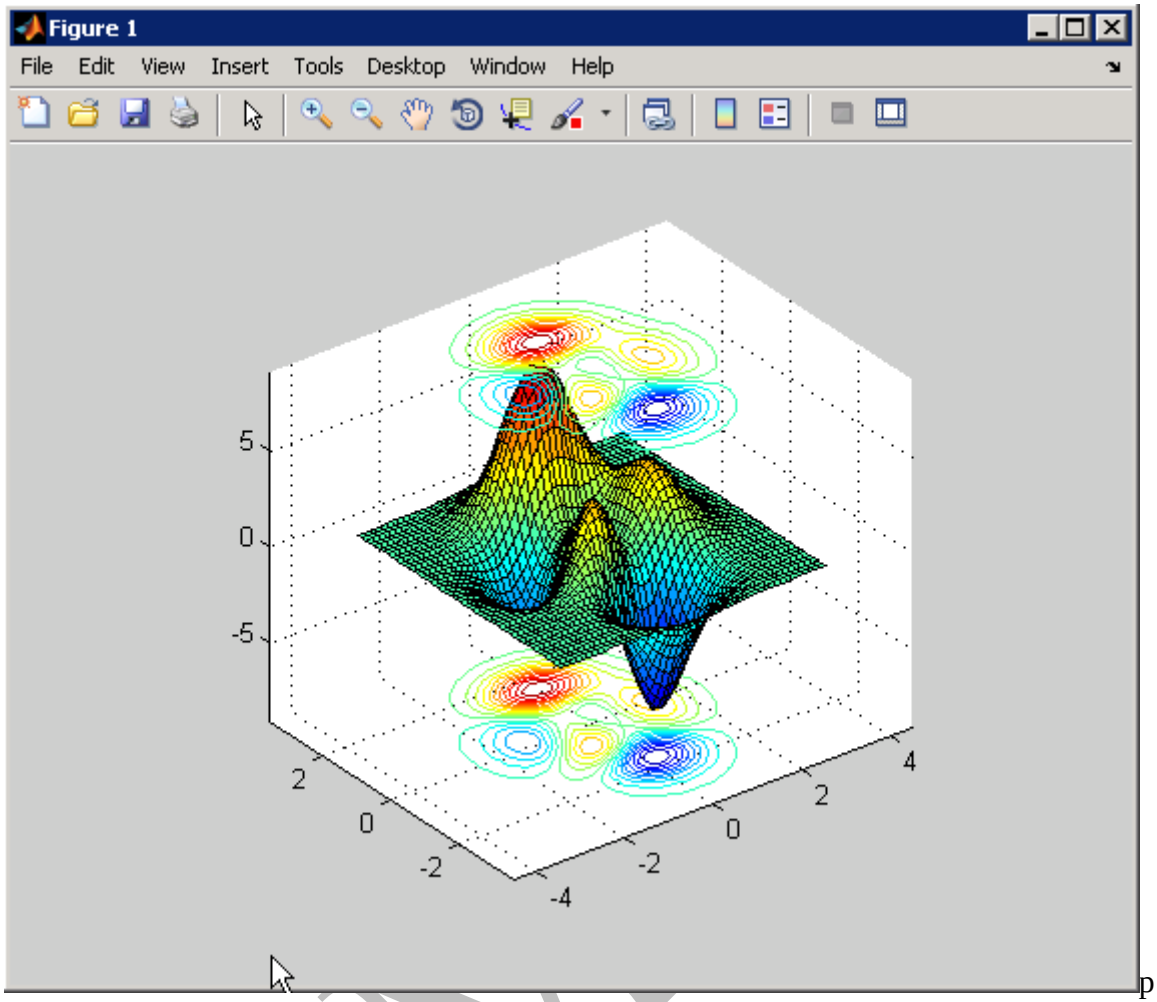
2D contour can also be used to facilitate the readiness of a 3D plot as you can see below:

EDITOR PUBLISH VIEW

```
1  -    [X,Y,Z] = peaks;
2  -    surf(X, Y, Z), hold on       % plot surface
3
4  -    ContoursNumber=20;
5  -    [~,h] = contour(X,Y,Z,ContoursNumber);   % plot contour at the bottom
6  -    hh = get(h, 'Children');
7  -    for i=1:numel(hh)
8  -        ZData = get(hh(i), 'XData');   % get matrix shape
9  -        ZData(:) = 9;              % fill it with constant Z value
10 -        set(hh(i), 'ZData',ZData);     % update patch
11 -    end
12
13 -    [~,h] = contour(X,Y,Z,ContoursNumber);   % plot contour at the top
14 -    hh = get(h, 'Children');
15 -    for i=1:numel(hh)
16 -        ZData = get(hh(i), 'XData');   % get matrix shape
17 -        ZData(:) = -9;             % fill it with constant Z value
18 -        set(hh(i), 'ZData',ZData);     % update patch
19 -    end
20
21 -    hold off
22 -    view(3); axis vis3d; grid on
```

That will give:

SAMP


Figure 1
File Edit View Insert Tools Desktop Window Help
5
0
-5
2
0
-2
-4
-2
0
2
4

p

SAM

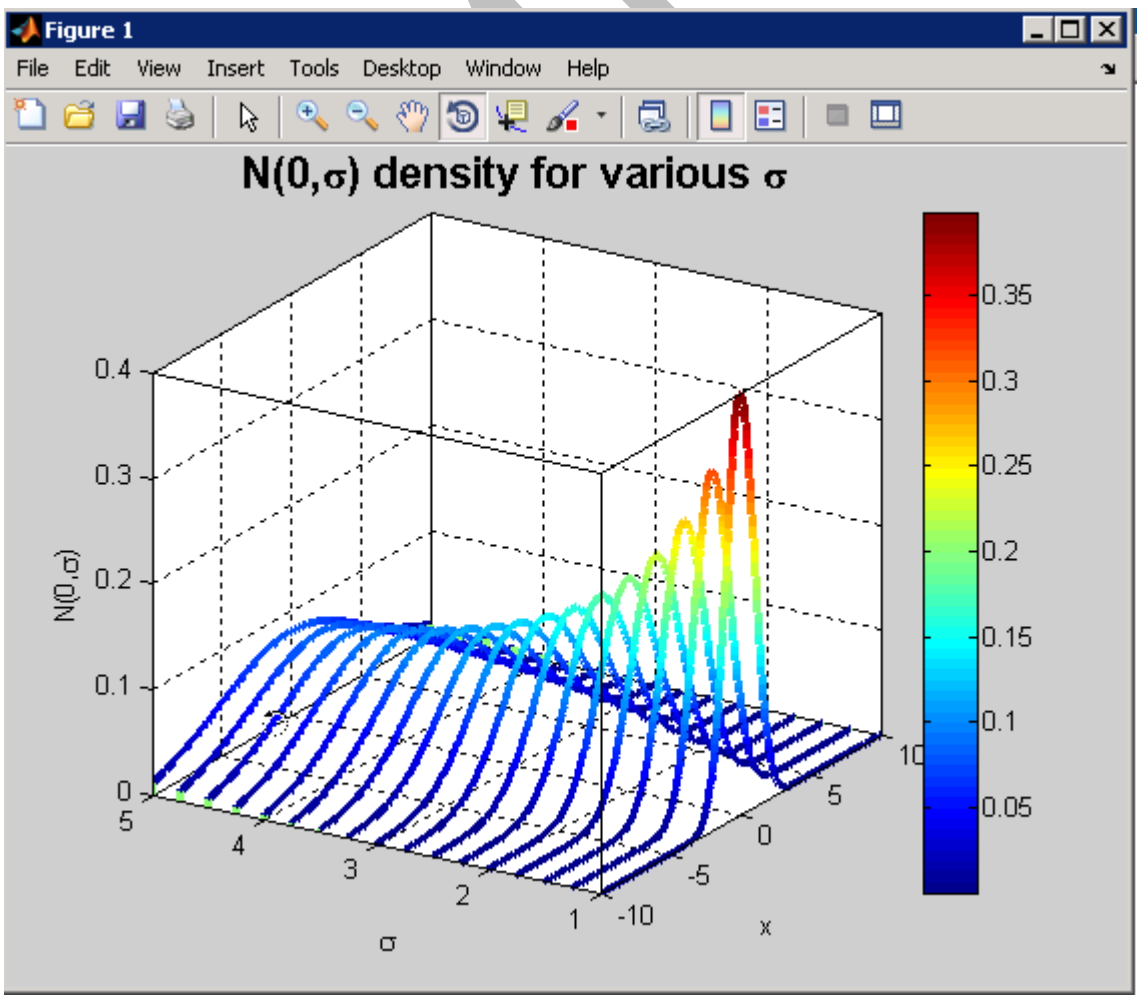
waterfall

The **waterfall( )** function is especially used in statistics to show the hypothesized density distribution function evolution of a regression or in finance to show how a distribution change with the time:

```
EDITOR  PUBLISH  VIEW

1 -  dsig = 0.25;dx = 0.05;mu = 0;
2 -  [X,SIGMA] = meshgrid(-10:dx:10,1:dsig:5);
3 -  Z = exp(-(X-mu).^2./(2*SIGMA.^2))./sqrt(2*pi*SIGMA.^2);
4 -  f=waterfall(X,SIGMA,Z)
5 -  set( f, 'LineWidth',3);
6 -  title('N(0,\sigma) density for various \sigma','fontweight','bold','fontsize',16)
7
8 -  box on;
9 -  xlabel('x');ylabel('\sigma');zlabel('N(0,\sigma)')
10 - rotate3d on;view(-58,24);
11 - grid on;
12 - colorbar;
13   %to see trough the grid
14 - hidden off;
```

The result will be:

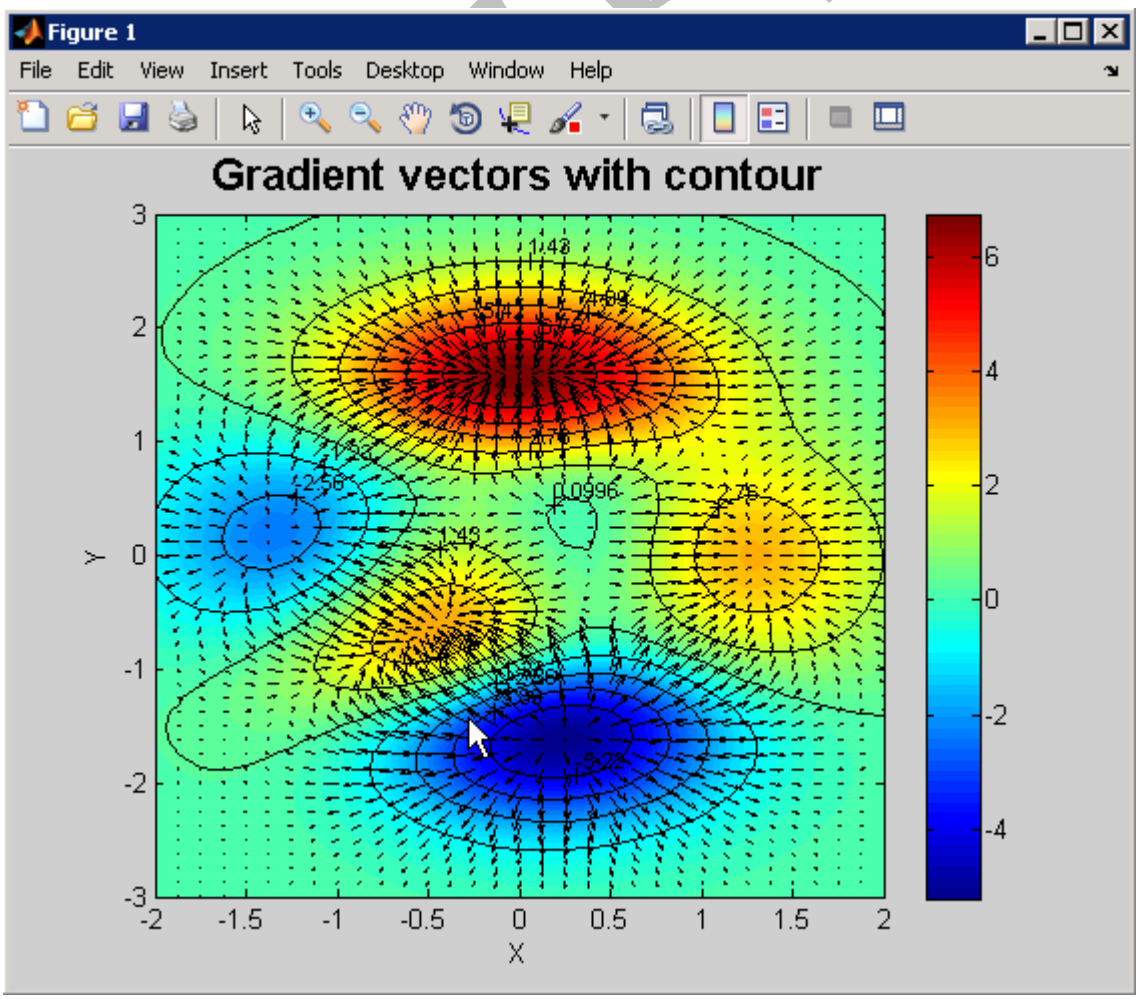


2D Gradient (quiver)

Gradient is mostly used by physicists in electrodynamics and various field implying gaz, liquids or plasmas. To obtain a gradient plot in MATLAB we have to mix the functions **gradient( )** and **quiver( )**:

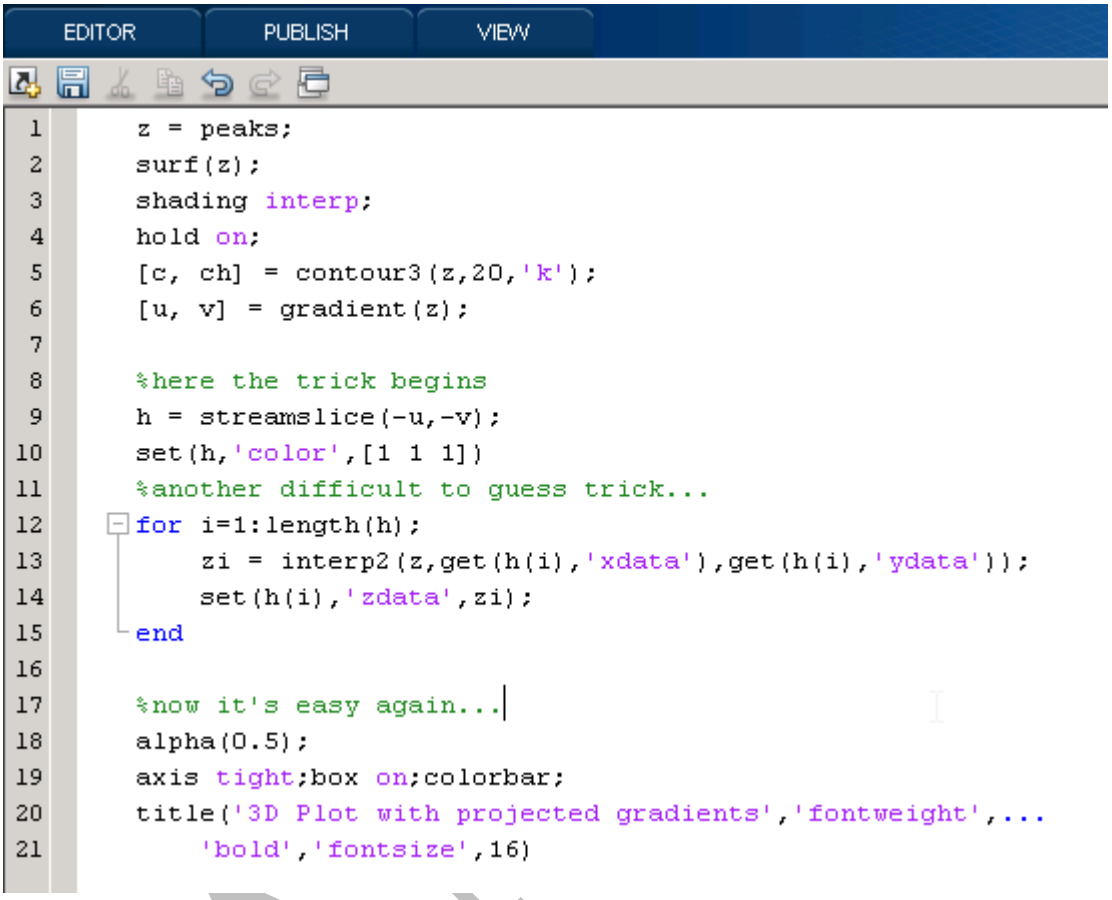
```
1   [X,Y,Z]=peaks;
2   [DX,DY]=gradient(Z);
3   [C,h]=contour(X,Y,Z,10,'k');
4   texth=clabel(C,'fontsize', 8);
5   hold on;
6   quiver(X,Y,DX,DY,2,'k');
7   pcolor(X,Y,Z);
8   shading interp;
9   colorbar;
10  axis([-2 2 -3 3]);
11  title('Gradient vectors with contour','fontweight','bold','fontsize',16);
12  xlabel('X');ylabel('Y');
```

The result will be:



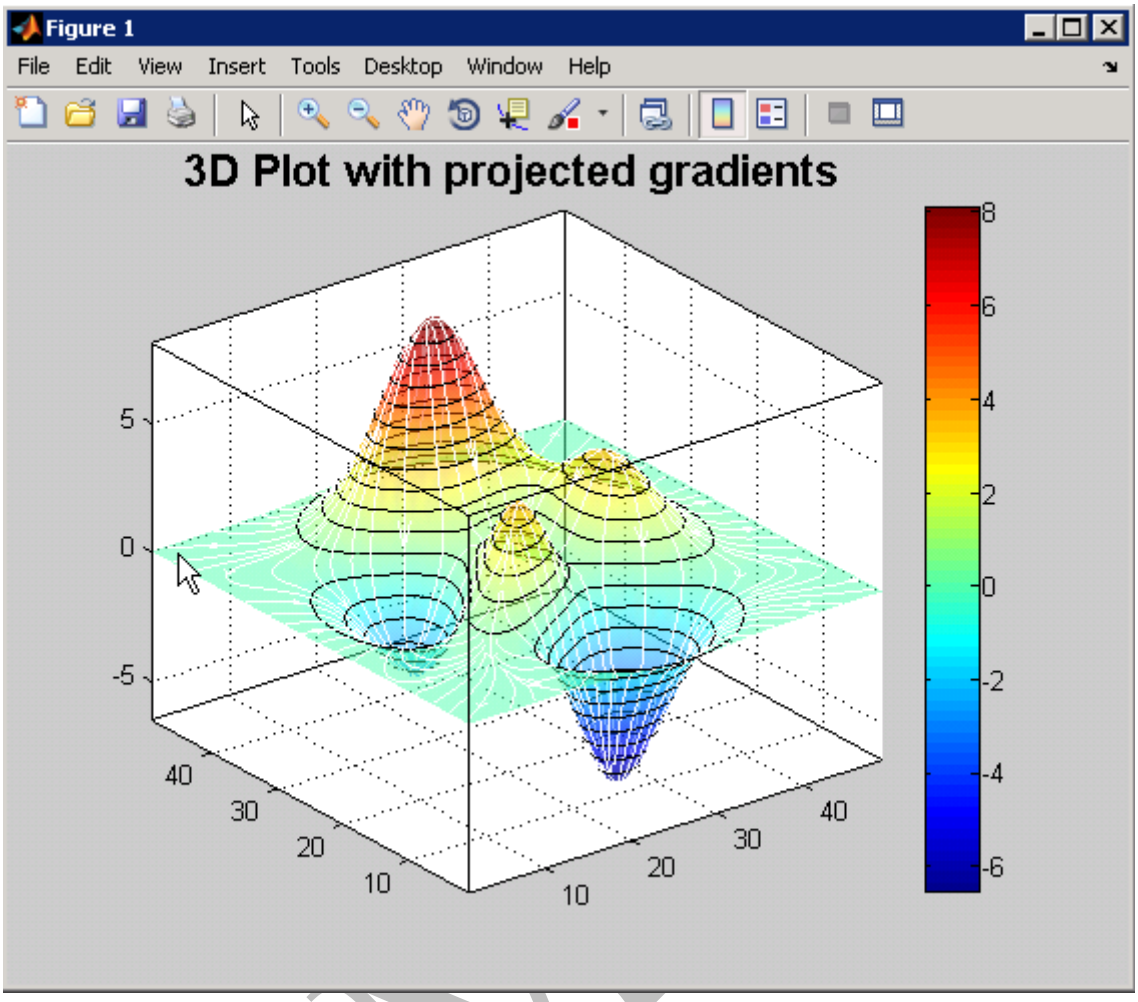
3D Gradient

A 3D gradient is most of time much more useful than a 2D one but the problem in MATLAB (as far as I know) is that there are no simple functions to get that. We need to apply a cook receipt using in a subtle way the **interp2( )** function to get the expected result



You can see the result on the next page (see the white lines that are the gradients perpendicular to the isoclines):

SA

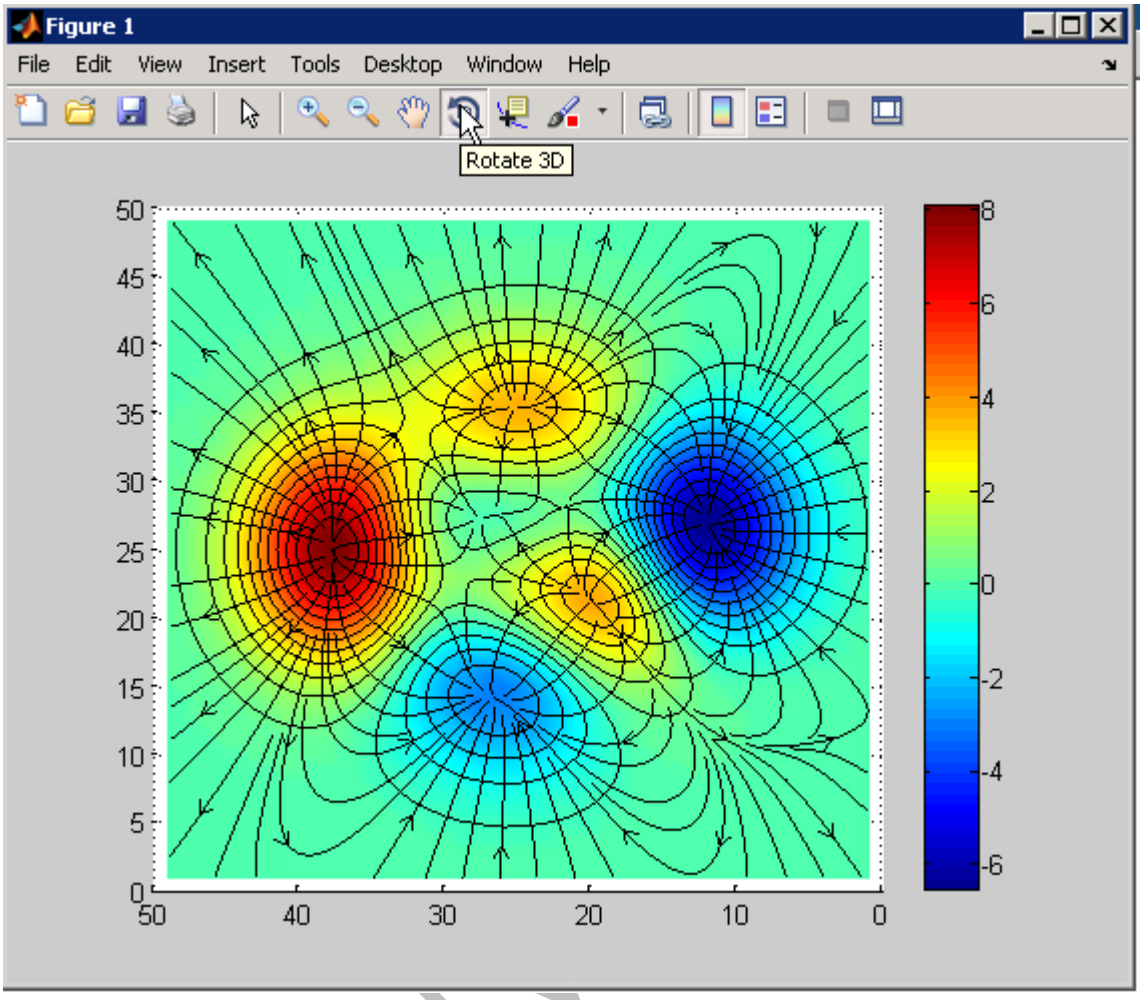


We can also play with the perspective:

EDITOR PUBLISH VIEW

```
1 -    z=peaks;
2 -    surf(z);
3 -    shading interp;
4 -    hold on;
5 -    [c,ch]=contour3(z,20,'k');
6 -    [u,v]=gradient(z);
7 -    h=streamslice(-u,-v);
8 -    set(h,'Color',[0 0 0]);
9 -    colorbar;
10
11 -   for k=1:length(h)
12 -       zk=interp2(z,get(h(k),'xdata'),get(h(k),'ydata'));
13 -       set(h(k),'zdata',zk);
14 -   end
15
16 -   rotate3d on;
17 -   view(-90,90);
```

To get:


Figure 1
File Edit View Insert Tools Desktop Window Help
Rotate 3D
50
45
40
35
30
25
20
15
10
5
0
50 40 30 20 10 0
8
6
4
2
0
-2
-4
-6

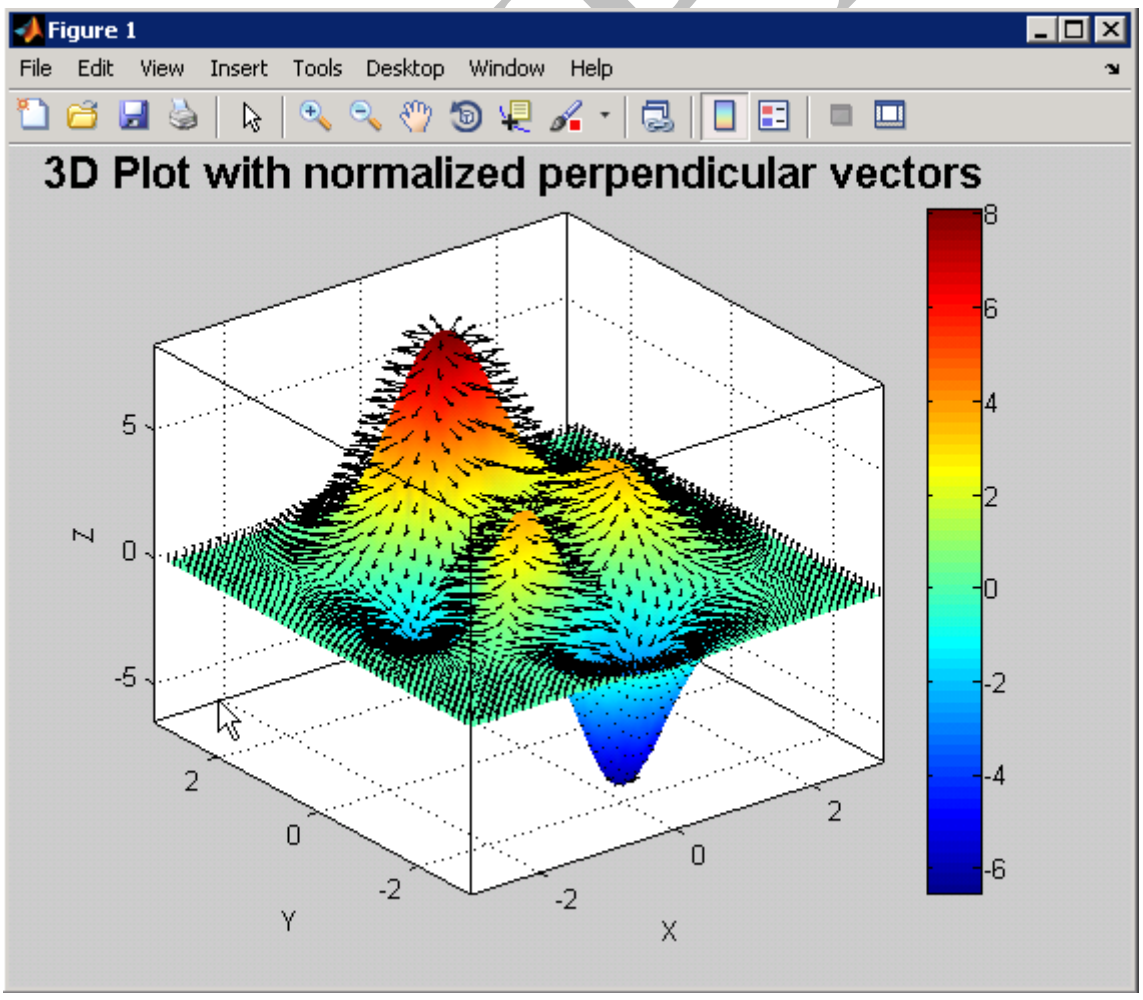
SAM

Perpendicular vectors

Until today I never saw anyone using this type of chart then I will just a simple example of the **surfnorm( )** function associated with **quiver3( )** this last one being used a lot in fluid mechanics:

```
1     [X,Y,Z]=peaks;
2     [U,V,W]= surfnorm(X,Y,Z);
3     surf(X,Y,Z)
4     hold('on')
5     quiver3(X,Y,Z, U,V,W, 1, 'k');
6     colorbar;
7     shading interp;
8     axis tight;box on;colorbar;
9     title('3D Plot with normalized perpendicular vectors','fontweight',...
10        'bold','fontsize',16)
11    xlabel('X');ylabel('Y');zlabel('Z');
```

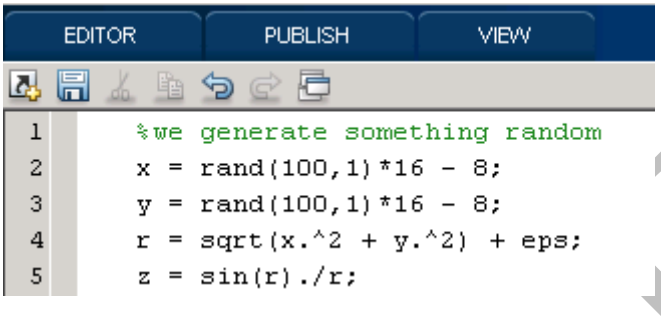
The result will be:



Surface plots of nonuniformly sampled data

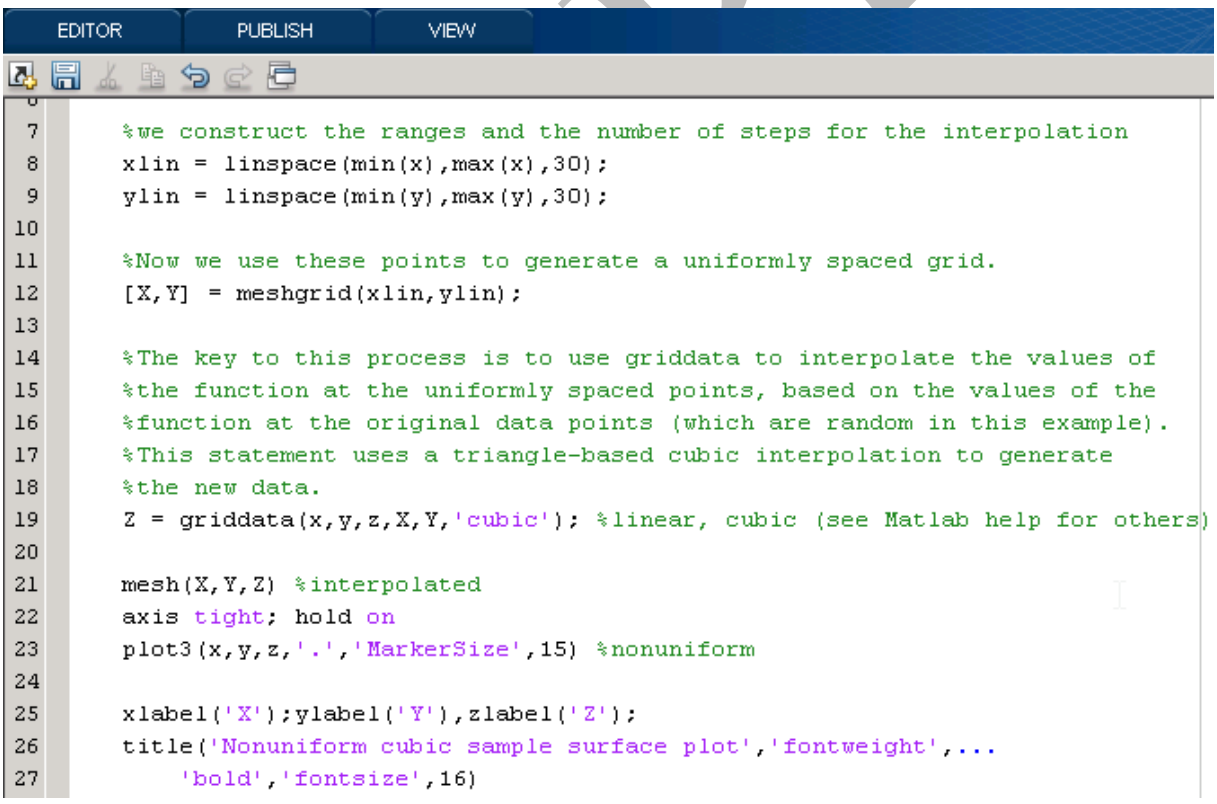
This is the most important example of 3D surface plot for people working in laboratories because most of time we have three vectors and for each $\{x, y\}$ we have a z.

Consider:

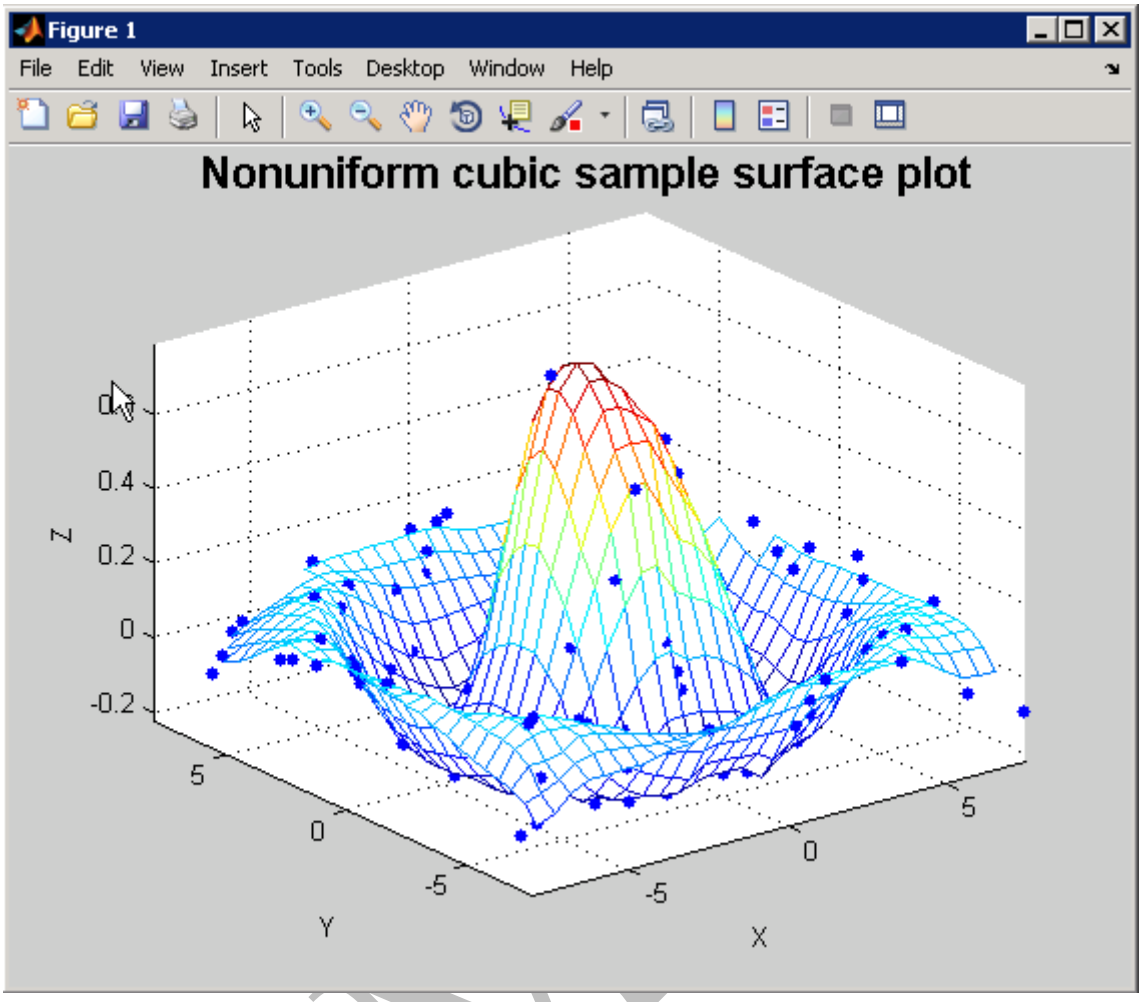


Cubic (triangle base interpolation)

The 1st method is to transform your non-uniform vectors into a uniform grid thanks to the combination of **linspace( )** and **meshgrid( )**. After the difficult trick is to use the function **griddata( )** to prepare a **mesh( )**. Finally we use a **plot3( )** to plot the points:



The result will looks like (see that we can have a grid that has more vertices than the original vectors sample points!):



Note: 'cubic' doesn't mean "(bi-)cubic spline"! if you edit the code of the function griddata you will be able to see that:

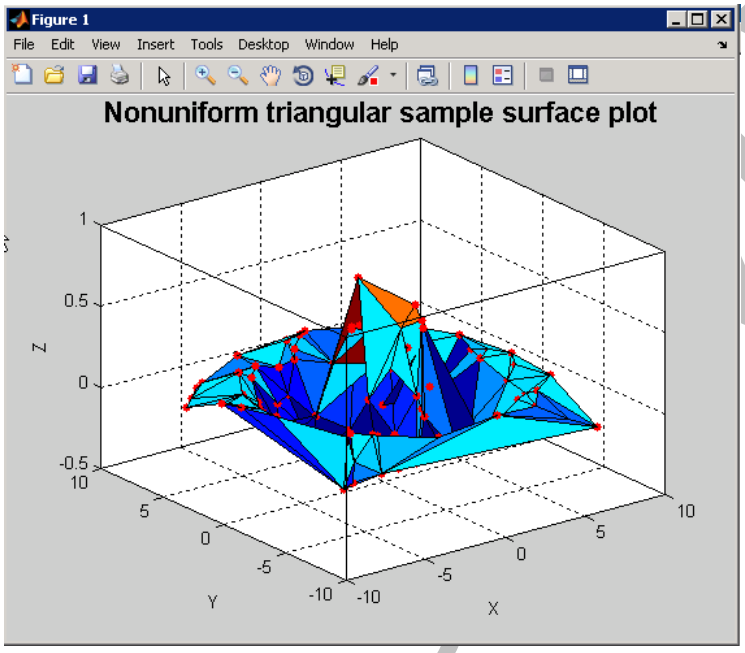
```
EDITOR  PUBLISH  VIEW
235     %-----------------------------------------------------------
236     function vq = cubic(x,y,v,xq,yq)
237     %TRIANGLE Triangle-based cubic interpolation
238
239     %   Reference: T. Y. Yang, "Finite Element Structural Analysis",
240     %   Prentice Hall, 1986.  pp. 446-449.
241     %
242     %   Reference: David F. Watson, "Contouring: A guide
243     %   to the analysis and display of spacial data", Pergamon, 1994.
244
245     % Triangulate the data
246
247 -   [x, y, v] = mergepoints2D(x,y,v);
248
249 -   dt = delaunayTriangulation(x,y);
250 -   scopedWarnOff = warning('off', 'MATLAB:triangulation:EmptyTri2DWarnId');
```

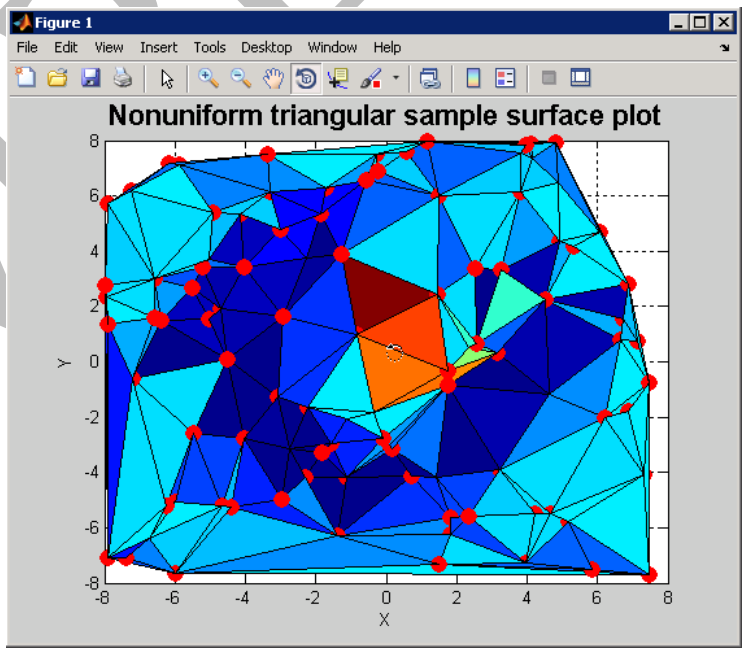
Delaunay (triangular connections) for open surfaces

The 2nd method below using the function **delaunay( )** and **trisurf( )** is easier to implement (shorter) but also vey difficult to guess:

```
29      p_h=plot3(x,y,z,'.r','MarkerSize',15);
30      hold on;grid on;box on;
31      tri=delaunay(x,y);
32      trisurf(tri,x,y,z);r
33
34      xlabel('X');ylabel('Y'),zlabel('Z');
35      title('Nonuniform triangular sample surface plot','fontweight',...
36          'bold','fontsize',16)
37
```

The result will be (see that the grid has the same number of vertices as the original vectors sample points!):





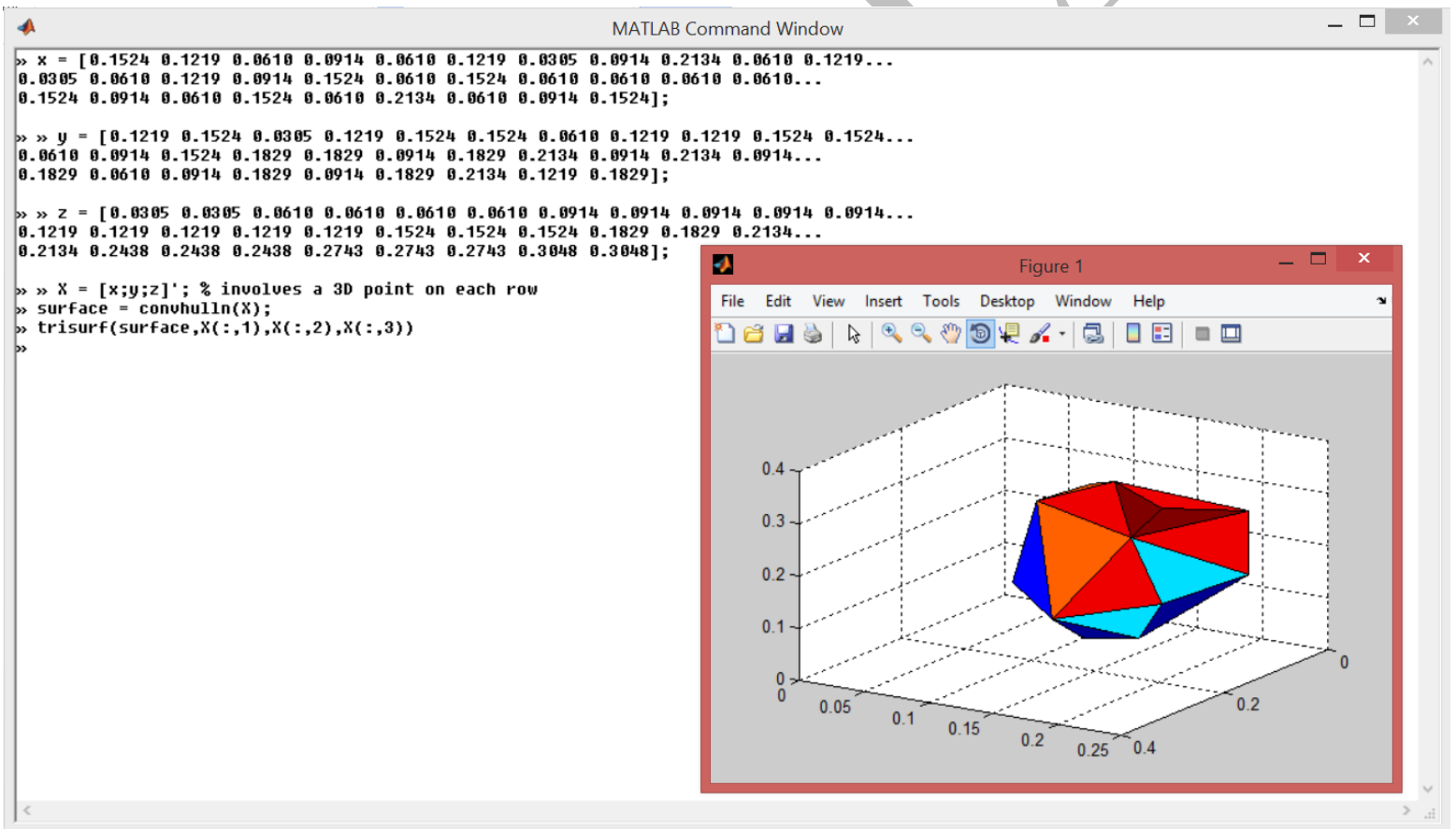
Hull convolution for closed surfaces

Sometimes we have the coordinates of points of a closed 3D surface and we need to search the best empirical method (algorithm) to plot the corresponding surface.

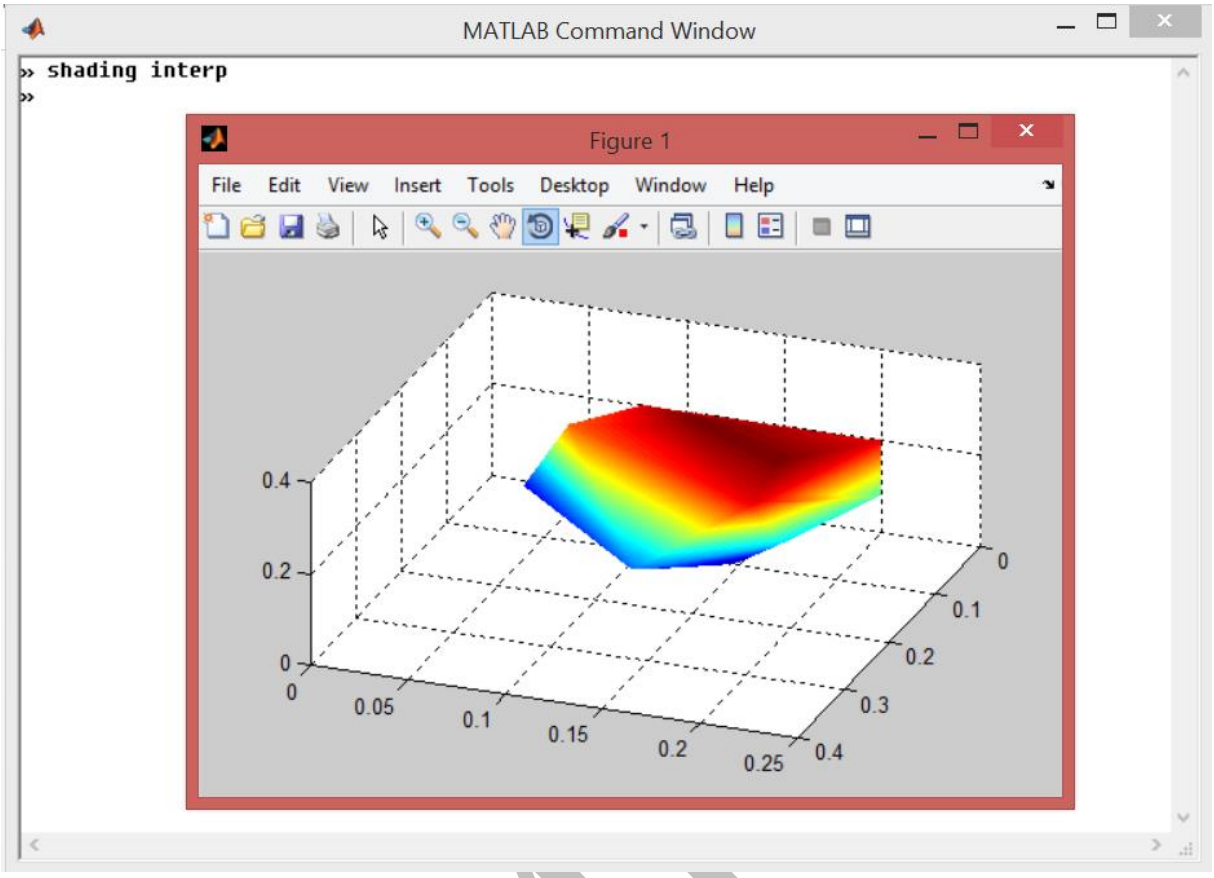
Consider for this purpose that we have the following coordinates:

```
x = [0.1524 0.1219 0.0610 0.0914 0.0610 0.1219 0.0305 0.0914 0.2134 0.0610 0.1219...
0.0305 0.0610 0.1219 0.0914 0.1524 0.0610 0.1524 0.0610 0.0610 0.0610 0.0610...
0.1524 0.0914 0.0610 0.1524 0.0610 0.2134 0.0610 0.0914 0.1524];
y = [0.1219 0.1524 0.0305 0.1219 0.1524 0.1524 0.0610 0.1219 0.1219 0.1524 0.1524...
0.0610 0.0914 0.1524 0.1829 0.1829 0.0914 0.1829 0.2134 0.0914 0.2134 0.0914...
0.1829 0.0610 0.0914 0.1829 0.0914 0.1829 0.2134 0.1219 0.1829];
z = [0.0305 0.0305 0.0610 0.0610 0.0610 0.0610 0.0914 0.0914 0.0914 0.0914 0.0914...
0.1219 0.1219 0.1219 0.1219 0.1219 0.1524 0.1524 0.1524 0.1829 0.1829 0.2134...
0.2134 0.2438 0.2438 0.2438 0.2743 0.2743 0.2743 0.3048 0.3048];
```

Using the functions **convhulln( )** and **trisurf( )** we get:



But what about smoothing the corresponding surface color instead of seeing the **trisurf( )** triangles?:


MATLAB Command Window
» shading interp
»
Figure 1
File Edit View Insert Tools Desktop Window Help
0.4
0.2
0
0
0.05
0.1
0.15
0.2
0.25
0.4
0.3
0.2
0.1
0

SAMPLE

Spline (biharmonic-spline)

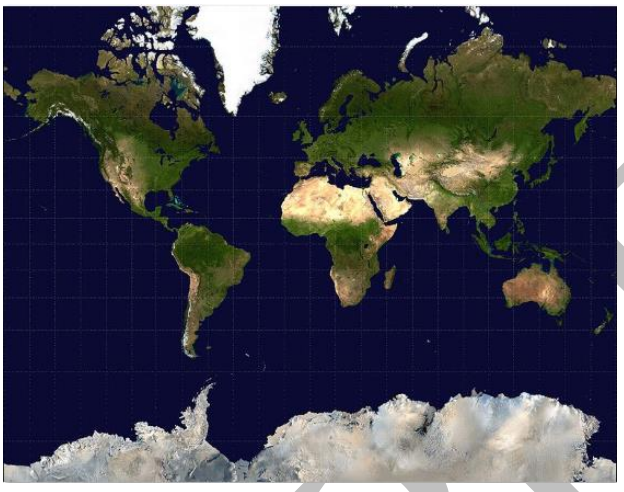
For an example see page 469.

SAMPLE

Image mapping plots

Map an image on a 2D plane chart

A typical use of mapping images in 2D plots is for statistical representations on any area in the World. For the following example, we will use the following image:



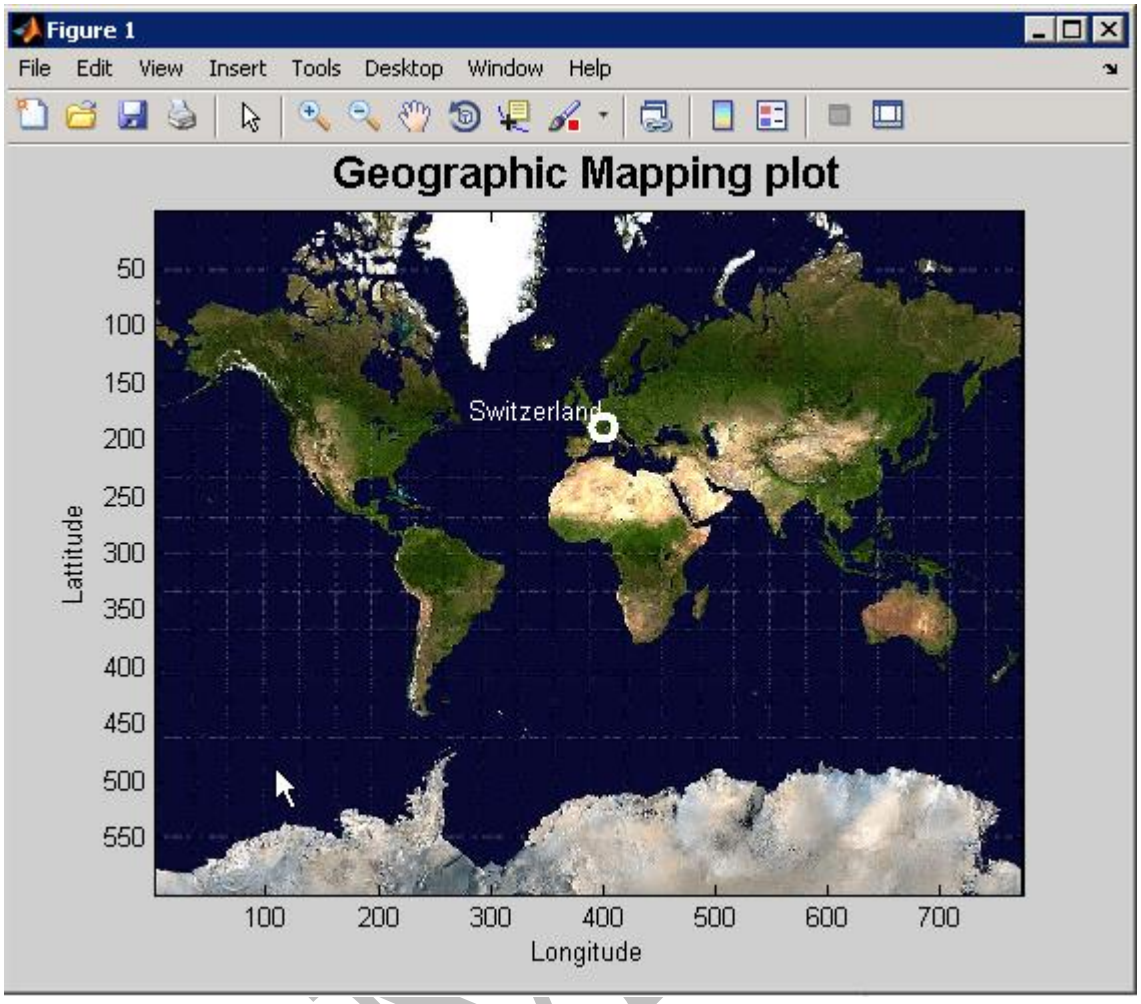
http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Mercator-projection.jpg/773px-Mercator-projection.jpg

After we write the following script using the functions **imread( )** and **imagesc( )**:

EDITOR PUBLISH VIEW

```
1     img = imread('http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Merc
2     x=400;y=190;
3     labels='Switzerland';
4     figure;
5     imagesc(img);
6     xlabel('Longitude');
7     ylabel('Lattitude');
8     hold on
9     plot(x,y, 'bo', 'MarkerSize',10, 'LineWidth',3,'color','white');
10    title('Geographic Mapping plot','fontweight',...
11        'bold','fontsize',16)
12    text(x,y, labels, 'Color','w', 'VerticalAlign',...
13        'bottom', 'HorizontalAlign','right')
```

The result is visible on next page:



Map an image on a 3D surface chart

A typical use of mapping images on 3D surfaces is for pedagogical purposes. Here is an well know example in General Relativity using the following image:

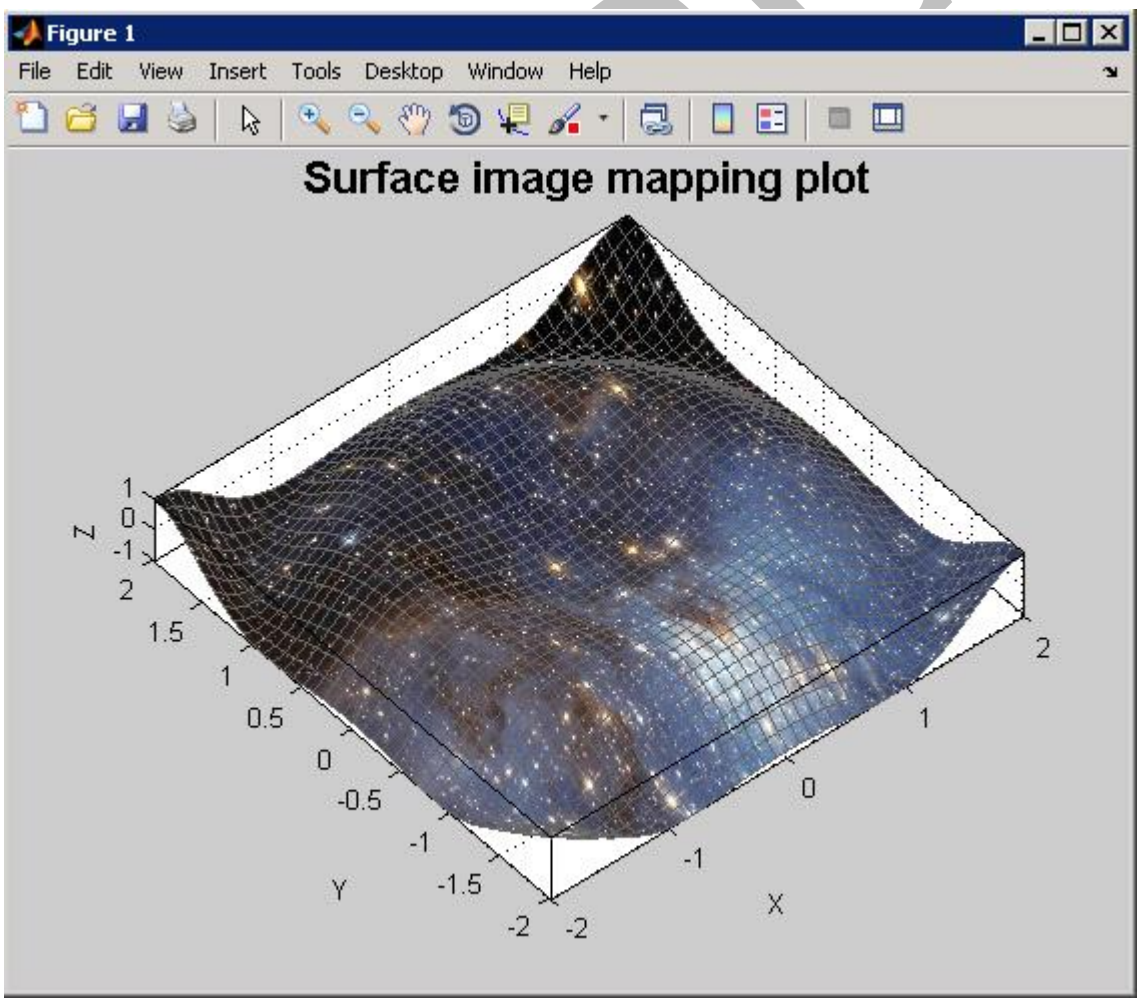


http://d1jqu7g1y74ds1.cloudfront.net/wp-content/uploads/2010/08/The-Brightest-of-Stars.jpg

The tip afterwards is to play with the fourth **texturemap** parameter of the function **surf( )**:

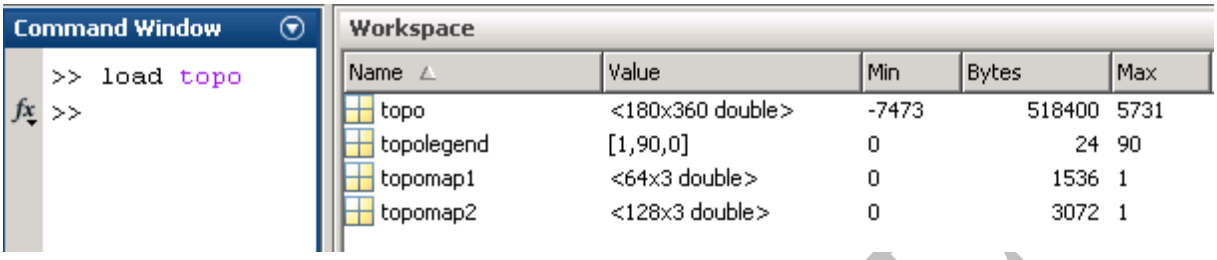
```
1   X=-2:0.1:2;
2   Y=X;
3   [x,y] = meshgrid(X, Y);
4   z = sin(x.^2 + y.^2);
5   a=imread('http://d1jqu7g1y74ds1.cloudfront.net/wp-content/uploads/2010/08/T
6   s=surf(x,y,z,a,'FaceColor','texturemap');
7   shading interp
8   set(s, 'EdgeColor', [0.4 0.4 0.4])
9   view([-40 82]);
10  box on;
11  xlabel('X');ylabel('Y'),zlabel('Z');
12  title('Surface image mapping plot','fontweight',...
13      'bold','fontsize',16)
```

The result will be:

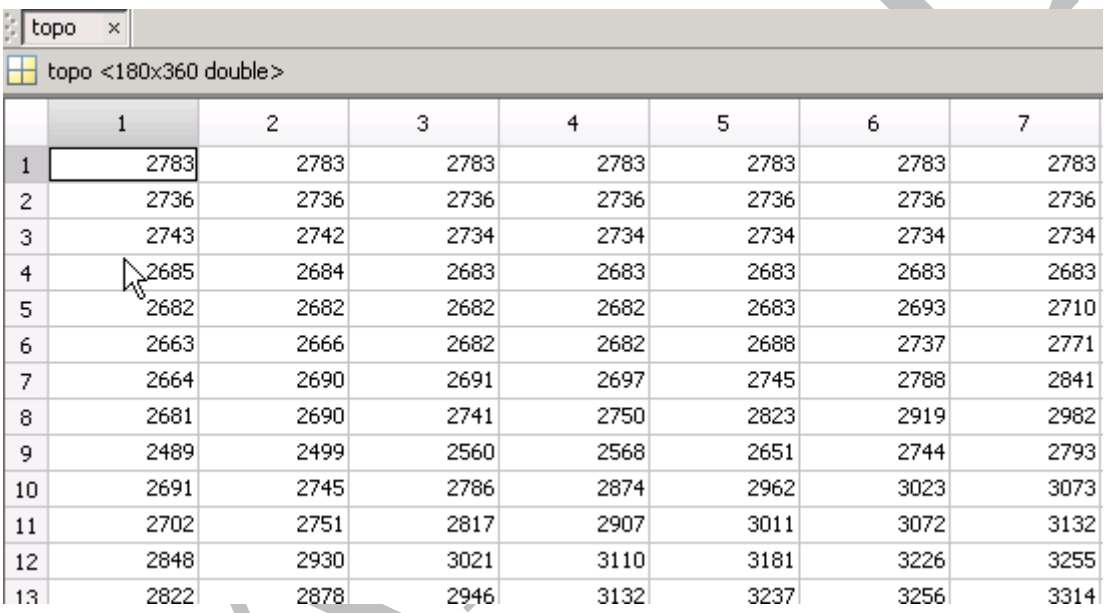


Sphere plot with data mapping and animated rotation

To be able the plot a sphere and map an image on it is especially useful for meteorologists, geophysicists and spatial engineer. For this example we will use a demo data mart of MATLAB:



The content of the variable *topo* looks as following:



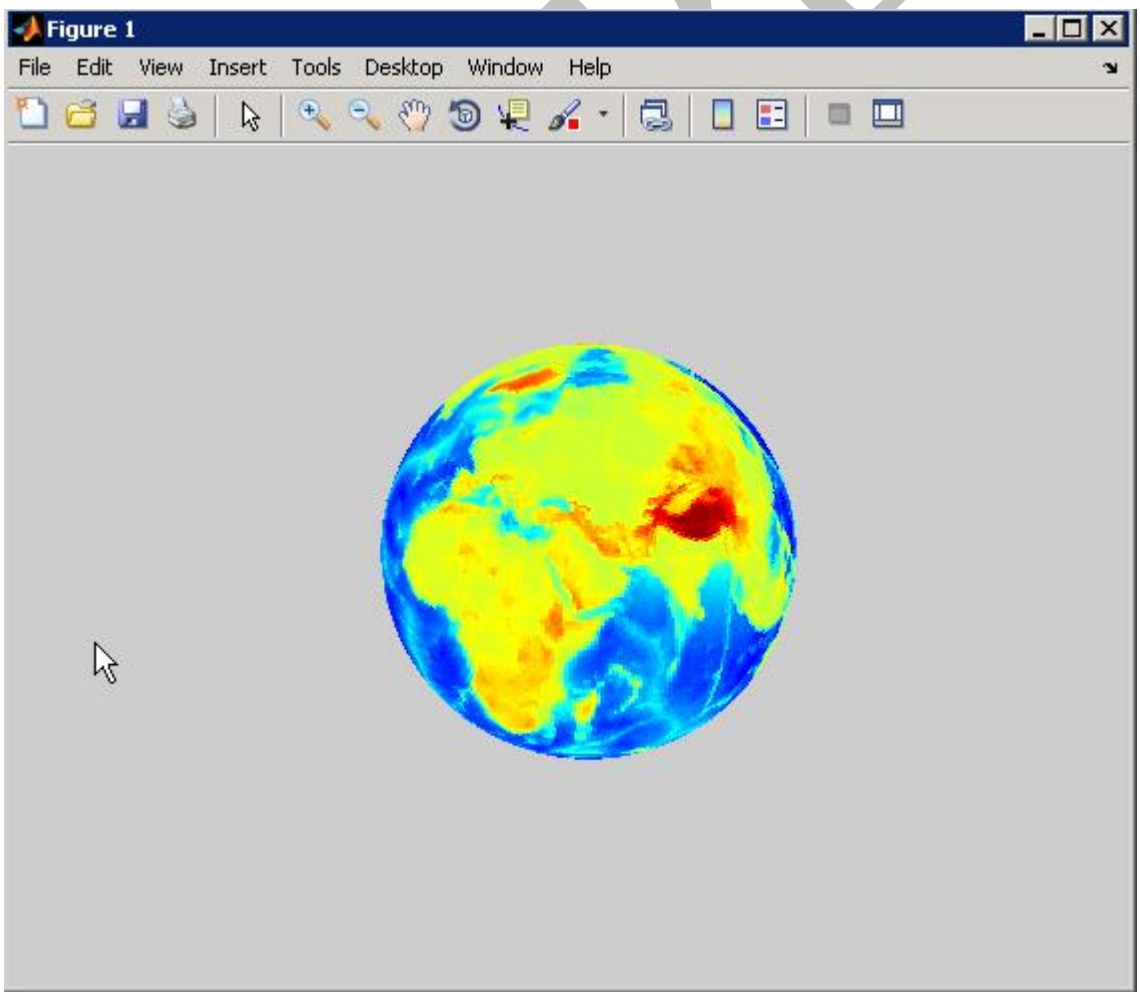
	1	2	3	4	5	6	7
1	2783	2783	2783	2783	2783	2783	2783
2	2736	2736	2736	2736	2736	2736	2736
3	2743	2742	2734	2734	2734	2734	2734
4	2685	2684	2683	2683	2683	2683	2683
5	2682	2682	2682	2682	2683	2693	2710
6	2663	2666	2682	2682	2688	2737	2771
7	2664	2690	2691	2697	2745	2788	2841
8	2681	2690	2741	2750	2823	2919	2982
9	2489	2499	2560	2568	2651	2744	2793
10	2691	2745	2786	2874	2962	3023	3073
11	2702	2751	2817	2907	3011	3072	3132
12	2848	2930	3021	3110	3181	3226	3255
13	2822	2878	2946	3132	3237	3256	3314

SAMPLE

Now let us plot using the functions **sphere( )** and **surface( )**:

```
1   n=300;
2   load topo;
3   [x,y,z]=sphere(n);
4   sh=surface(x,y,z,...
5          'facecolor','texturemap',...
6          'cdata',topo,'EdgeColor','none','LineStyle','none');
7
8   axis off;axis equal;
9   view(3); %see face
10  for i=1:20 %you can change the "20" to the value you want
11       rotate(sh,[0,0,1],1);
12       pause(1);
13  end
```

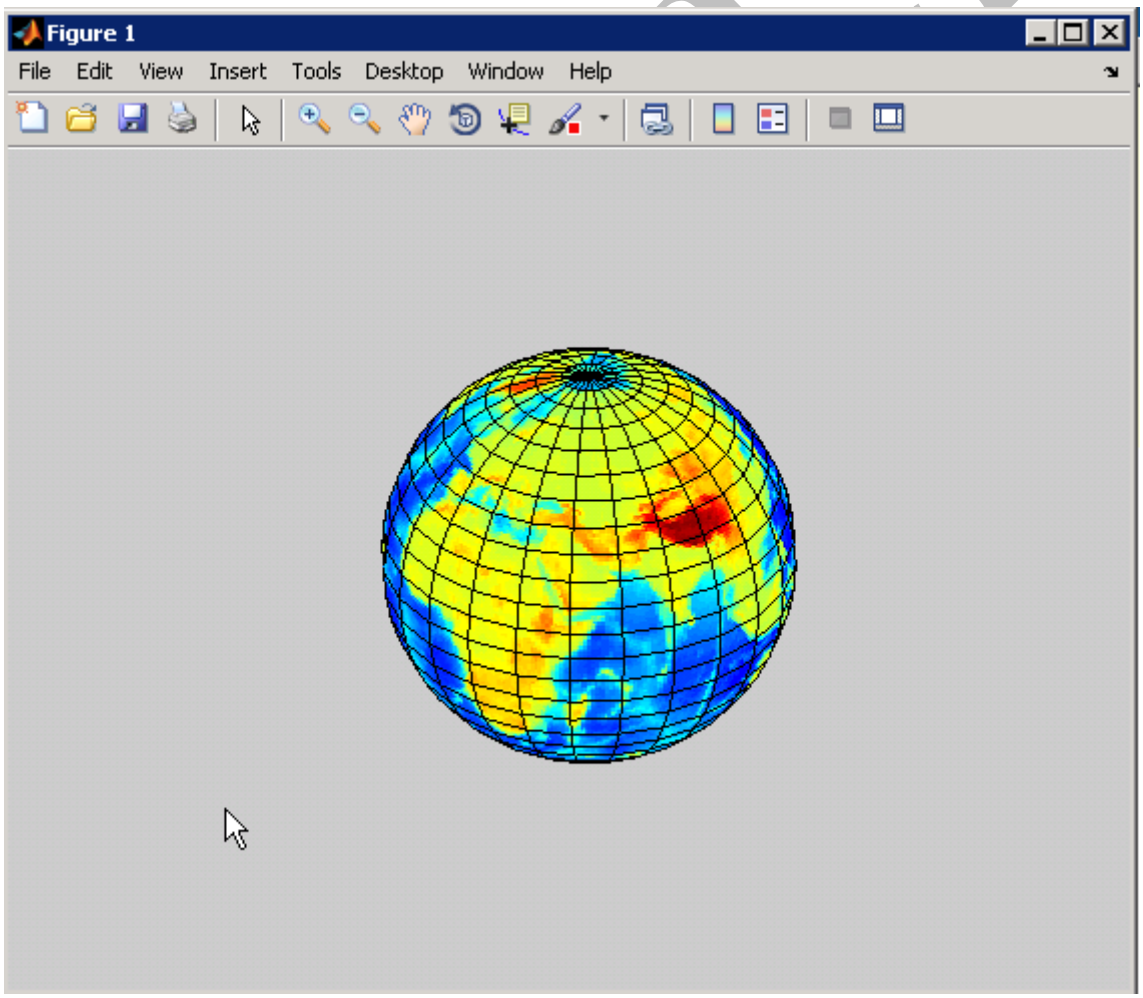
This is how the result will look like:



Now to have something more "geographic" let us draw a sphere with 24 meshes lines:

```
1   n=24;
2   load topo;
3   [x,y,z]=sphere(n);
4   sh=surface(x,y,z,'facecolor','texturemap','cdata',topo);
5   axis off;axis equal;
6   view(3); %see face
7   for i=1:20 %you can change the "20" to the value you want
8       rotate(sh,[0,0,1],1);
9       pause(1);
10  end
```

This will give:



Nice isn't it ;-)

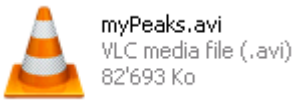
Export animated plots as *.avi or *.gif

*Export to *.avi*

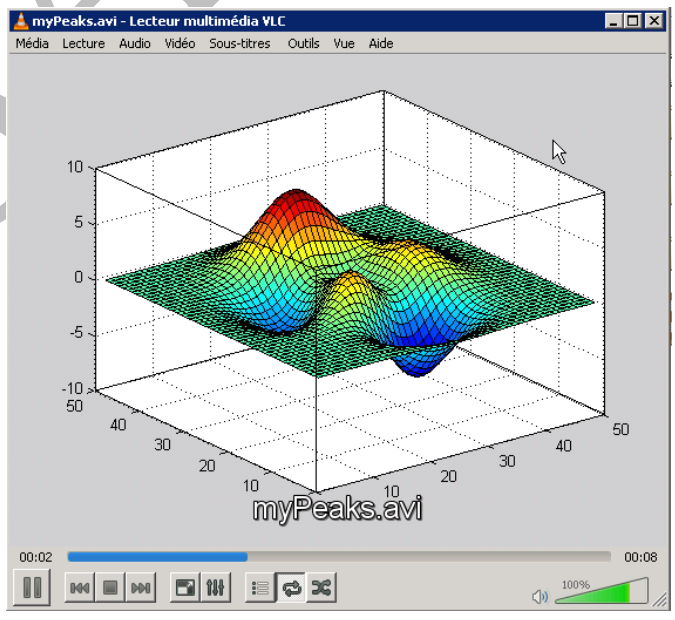
It is very useful for PowerPoint presentation to export as *.avi or for TV documentary purposes. For this, we use the function **getframe( )** and **movie2avi( )**:

```
%Animated chart with *.avi as output
clear all;close all;clc
nFrames = 120;
mov(1:nFrames) = struct('cdata', [],'colormap', []);
Z = peaks; surf(Z);
% Keep axes constant
axis([0 50 0 50 -10 10])
box on;
set(gca,'nextplot','replacechildren');
for k = 1:nFrames
    surf(sin(2*pi*k/nFrames)*Z,Z)
    mov(k) = getframe(gcf);
end
movie2avi(mov, 'c:\myPeaks.avi', 'compression', 'None');
```

The result will create a file on your hard drive:



And when you will open it you will get a high quality video:



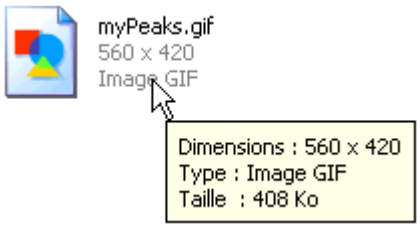
*Export to *.gif*

Export a chart animation as a *.gif is very useful for PowerPoint presentations and especially for Web pages.

The example below use the function **struct( )**, **getframe( )**, **frame2im( )**, **rgb2im( )** and **imwrite( )** to export and animated **peaks** plot:

```
1   %Animated *.gif output (most used case9
2   clear all;close all;clc
3   nFrames = 20;
4   Z=peaks;surf(Z);
5   box on;
6   axis([0 50 0 50 -10 10])
7   set(gca,'NextPlot','replaceChildren');
8   % Preallocate the struct array for the struct returned by getframe
9   F(nFrames) = struct('cdata',[],'colormap',[]);
10  % Record the movie
11  for j = 1:nFrames
12      surf(.01+sin(2*pi*j/nFrames)*Z,Z)
13      F(j) = getframe;
14      %the next lines are to add the frame to the gif (difficult to guess...)
15      F = getframe(gcf); %first we get the frame from the chat
16      RGB = frame2im(F); %we convert the frame in true color
17      [IND,map] = rgb2ind(RGB, 255); %we convert in 255 indexed colors gif
18      outfile = 'c:\myPeaks.gif';
19      if j==1
20          imwrite(IND,map,outfile,'gif','DelayTime',0,'loopcount',inf);
21      else
22          imwrite(IND,map,outfile,'gif','DelayTime',0,'writemode','append');
23      end
24  end
```

This will create *.gif a file that you can insert on your webpage or PowerPoint presentation:

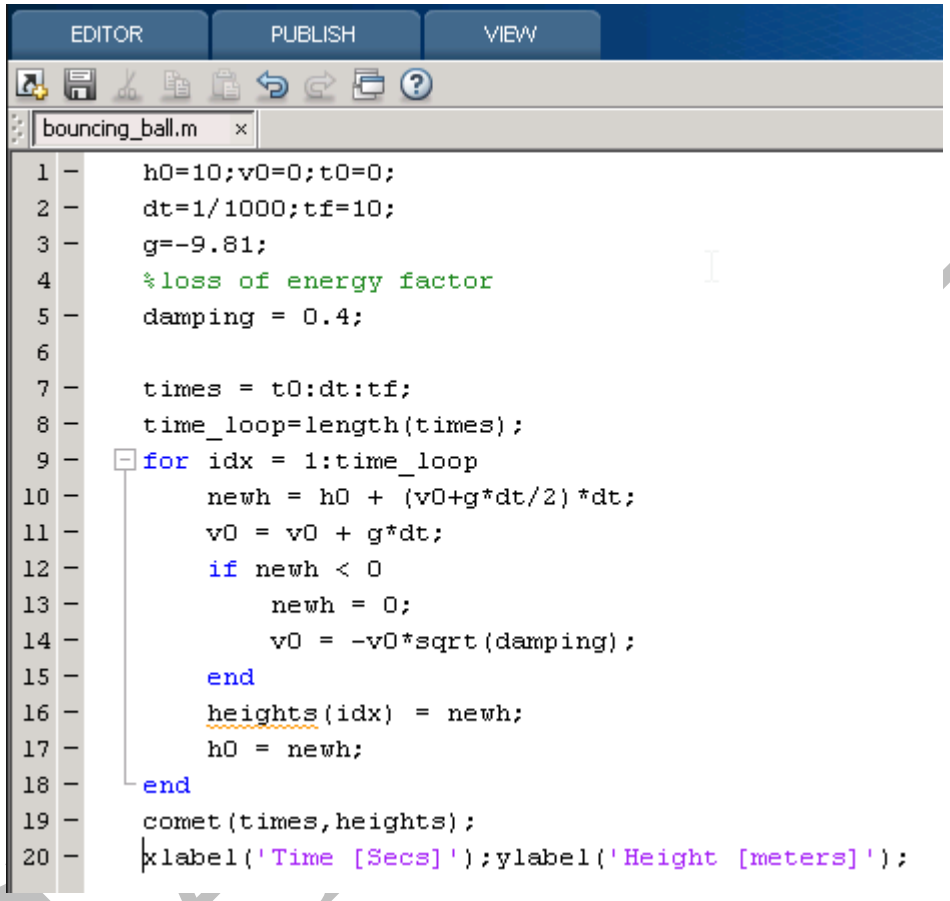


Engineering/Simulations

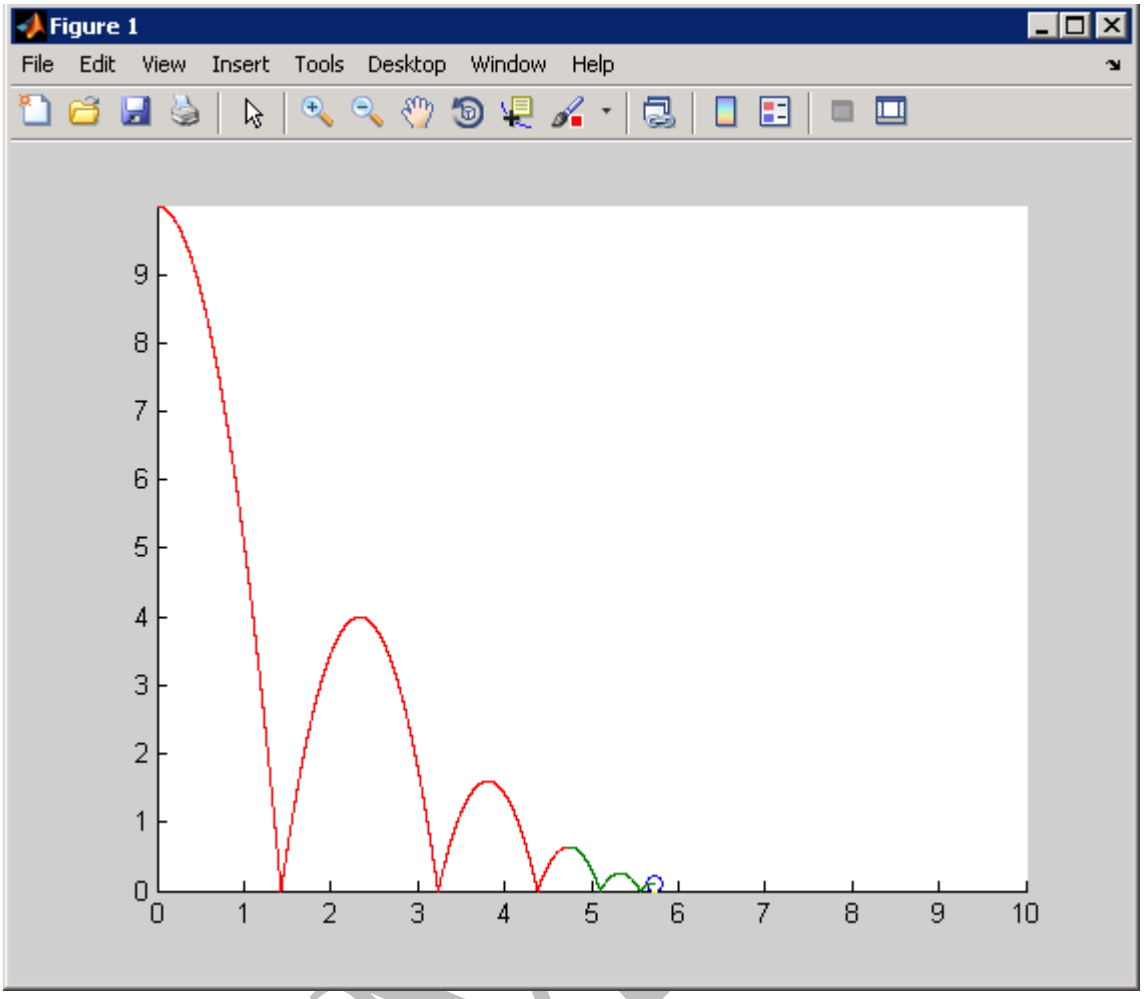
The purpose of this section is to apply some concepts we have studied in details in various theoretical courses.

Bouncing ball

Typical high school subject and very important to introduce students at simulation principles:



This will give:



Lagrange points potential surface

This is a well know example in celestial mechanics and very useful for some famous spatial missions. No commands will be new here, because the only purpose is just to plot what we have proved in the theoretical course as we have mention before.

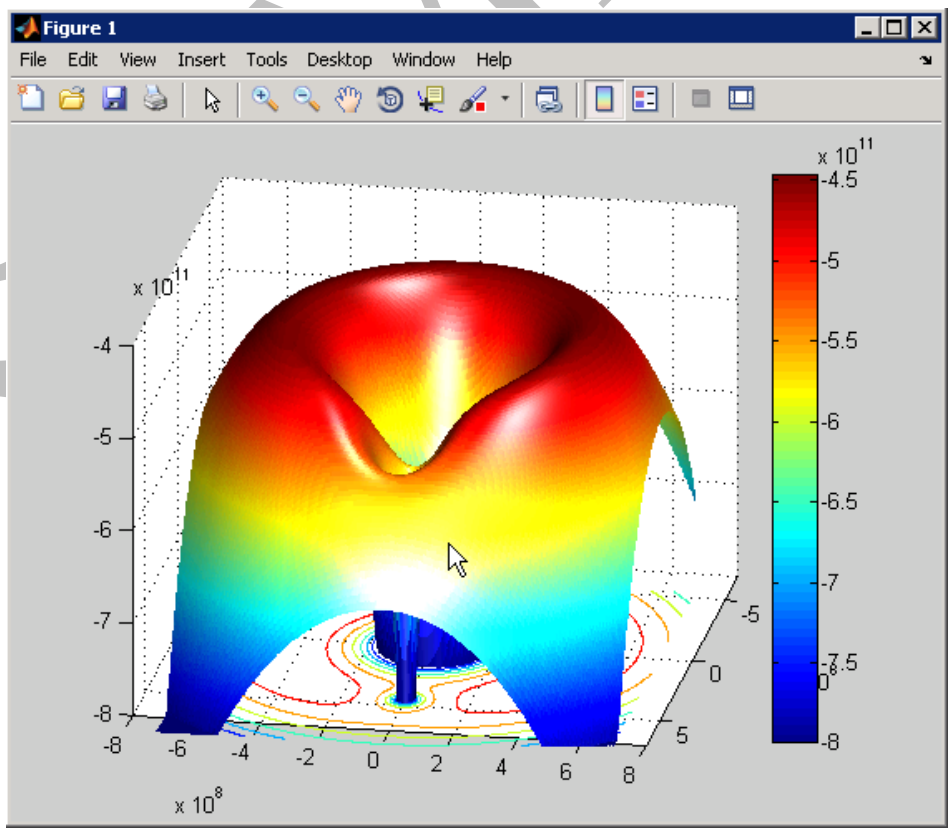
Also consider the following script:

SAMPLE

EDITOR PUBLISH VIEW

```
x1=linspace(-7E8,-8E5,150);
x2=linspace(8E5,1.2E8,150);
x3=linspace(1.6E8,7E8,150);
x=x1+x2+x3;
y=linspace(-7E8,7E8,450);
[X,Y]=meshgrid(x,y);
f=@(x,y)-(1.3346E20)./(sqrt((x-450).^2+y.^2))-(1.0038E19)./(sqrt((x-449999550).^2+y.^2))...
-(6.9E-7.*(x.^2+y.^2));
z=f(X,Y);
for i=1:450;
    for j=1:150;
        if (z(i,j)<-0.8E12)
            z(i,j)=-0.8E12;
        end;
     end;
end;
contour(X,Y,z,100);
mesh(X,Y,z);
meshc(X,Y,z);
az=100;el=25;view(az,el);
axis([-7E8 7E8 -0.8E9 0.8E9  -8E11 -4E11]);
colorbar;light;camlight('right');
```

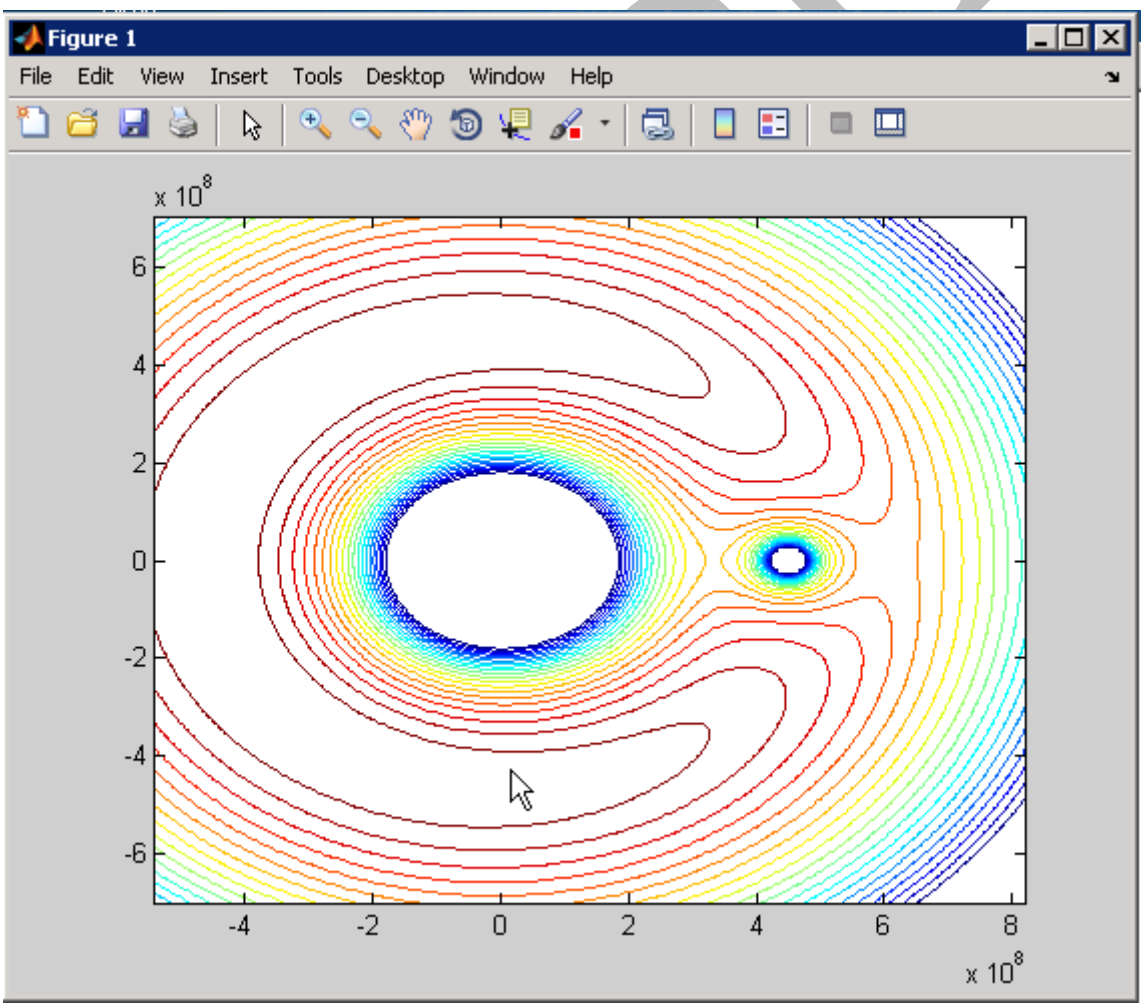
So we get (see next page):



And if we execute only this part of the script:

```
x1=linspace(-7E8,-8E5,150);
x2=linspace(8E5,1.2E8,150);
x3=linspace(1.6E8,7E8,150);
x=x1+x2+x3;
y=linspace(-7E8,7E8,450);
[X,Y]=meshgrid(x,y);
f=@(x,y)-(1.3346E20)./(sqrt((x-450).^2+y.^2))-(1.0038E19)./(sqrt((x-449999550).^2+y.^2))...
-(6.9E-7.*(x.^2+y.^2));
z=f(X,Y);
for i=1:450;
    for j=1:150;
        if (z(i,j)<-0.8E12)
            z(i,j)=-0.8E12;
        end;
    end;
end;
contour(X,Y,z,20);
```

We get:

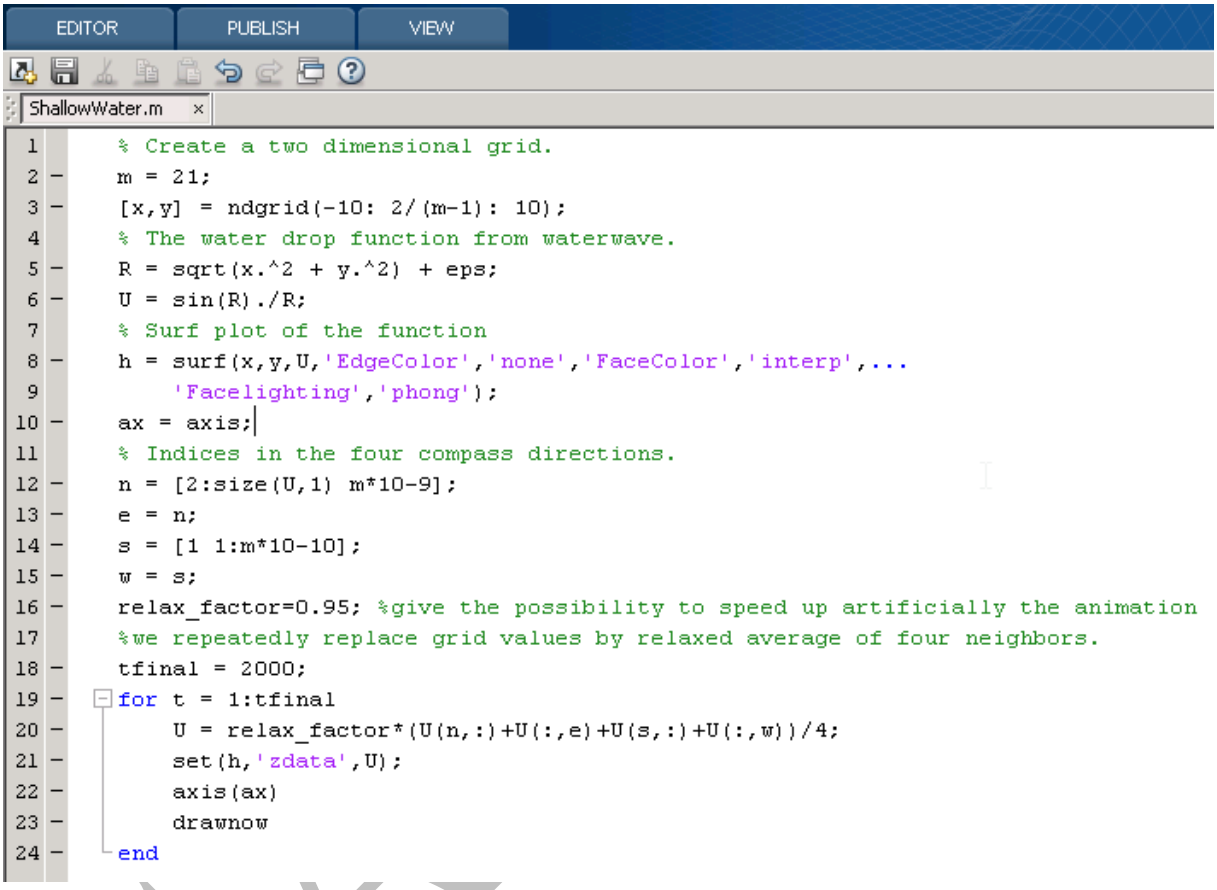


Finite Element Method

FEM Mesh simulation by simple average

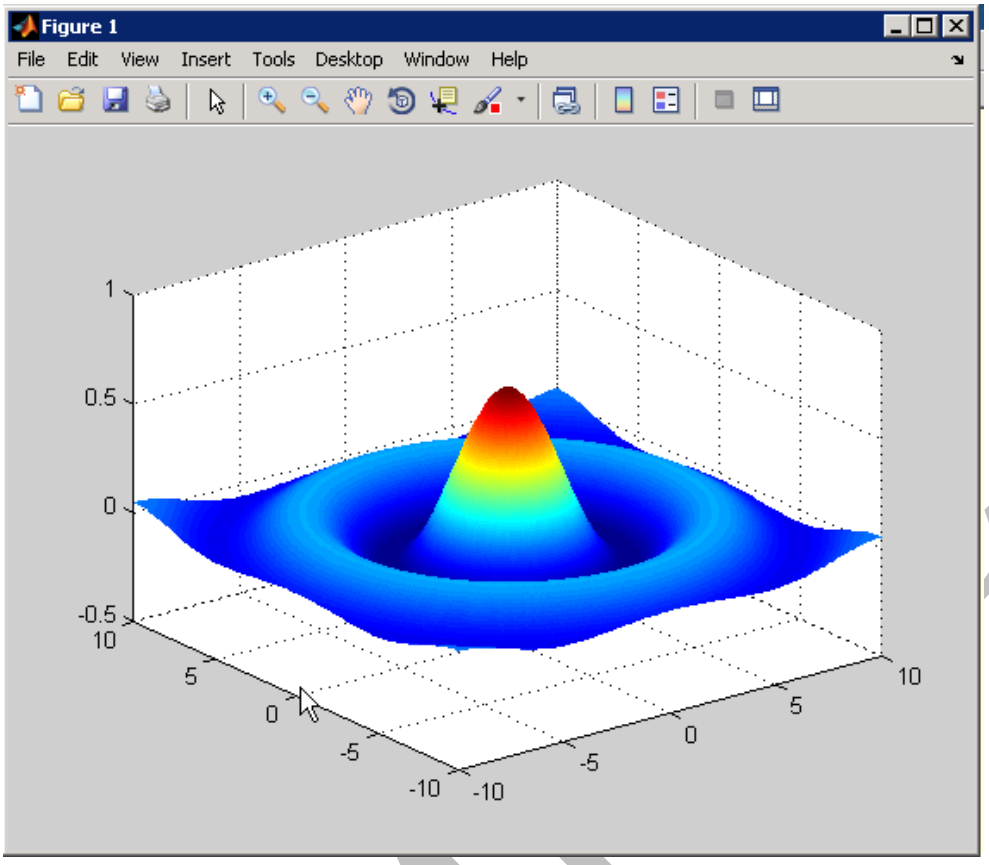
The purpose of this example is to introduce the most simple FEM (Finite Elements Method) simulation method using simple average.

Consider the following high-school case:

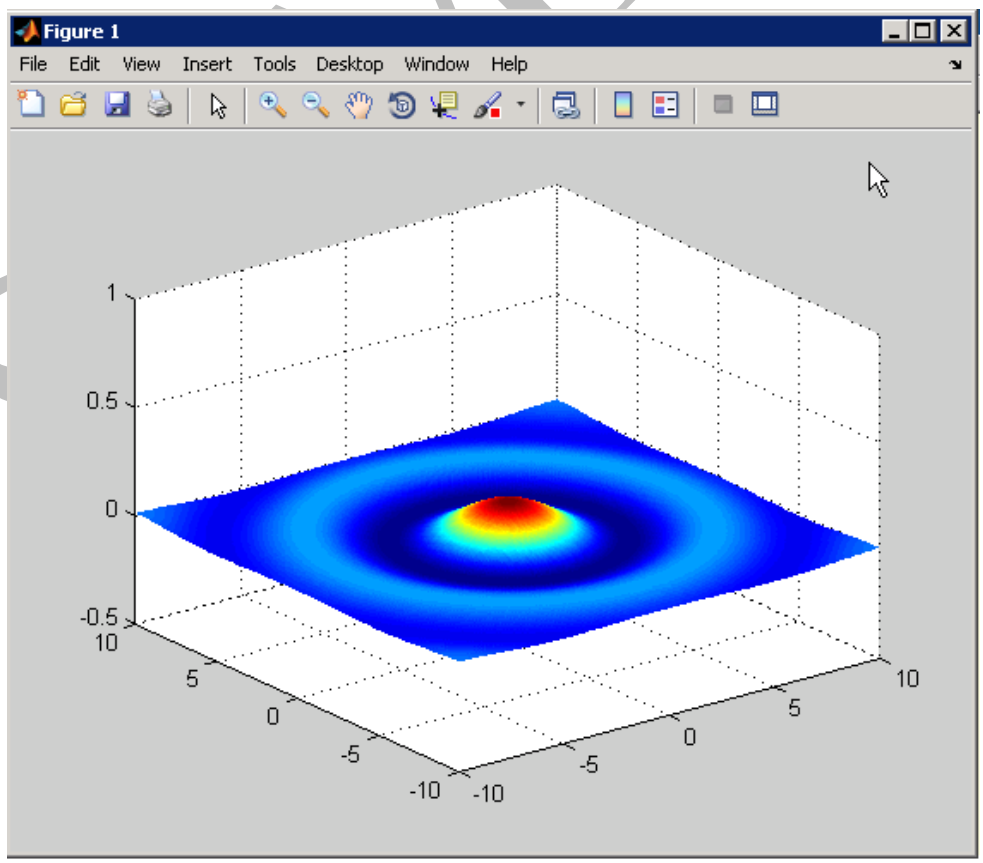



```
% Create a two dimensional grid.
m = 21;
[x,y] = ndgrid(-10: 2/(m-1): 10);
% The water drop function from waterwave.
R = sqrt(x.^2 + y.^2) + eps;
U = sin(R)./R;
% Surf plot of the function
h = surf(x,y,U,'EdgeColor','none','FaceColor','interp',...
    'Facelighting','phong');
ax = axis;
% Indices in the four compass directions.
n = [2:size(U,1) m*10-9];
e = n;
s = [1 1:m*10-10];
w = s;
relax_factor=0.95; %give the possibility to speed up artificially the animation
%we repeatedly replace grid values by relaxed average of four neighbors.
tfinal = 2000;
for t = 1:tfinal
    U = relax_factor*(U(n,:)+U(:,e)+U(s,:)+U(:,w))/4;
    set(h,'zdata',U);
    axis(ax)
    drawnow
end
```

This will give:



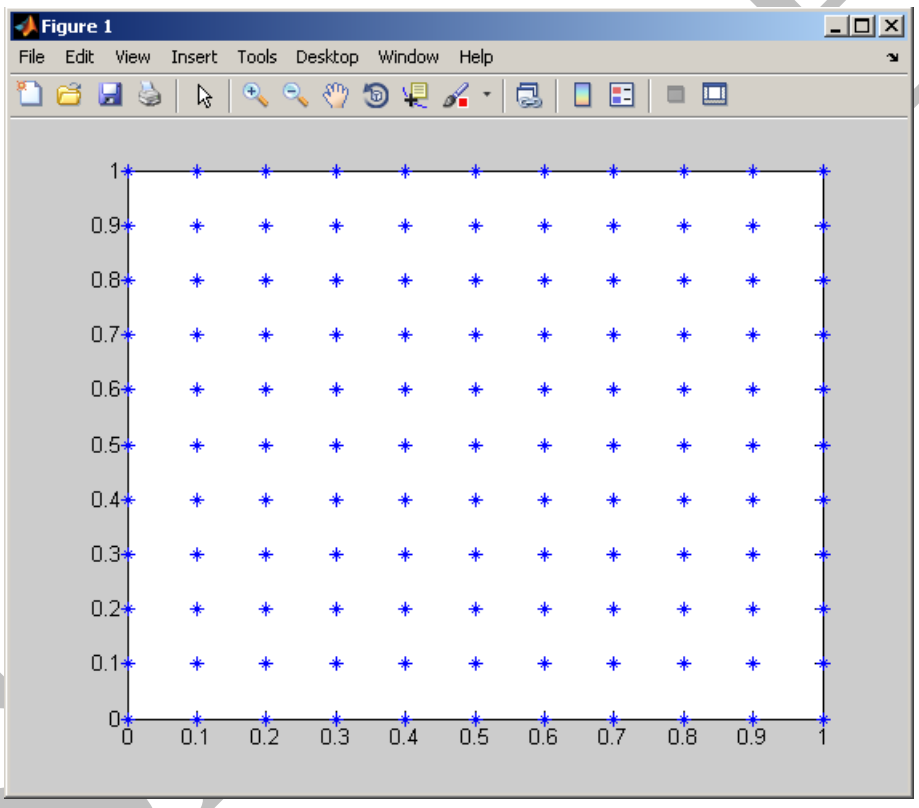
And after a few seconds:



FEM Building 2D regular square grid

GridPoint2D.m* | GridPoint3D.m

```
1  clear;clc
2  %Absolute Reference Mesh Input Data
3  x0=0; x00=1;
4  y0=0; y00=1;
5  lx=x00-x0;
6  ly=y00-y0;
7  M=10; N=10;
8  dx=lx/M; dy=ly/N;
9  % Matrix Mesh Generation matlab Command
10 [x,y]=meshgrid(x0:dx:x00,y0:dy:y00);
11 % Output Data Visualization
12 plot(x,y,'*b')
```



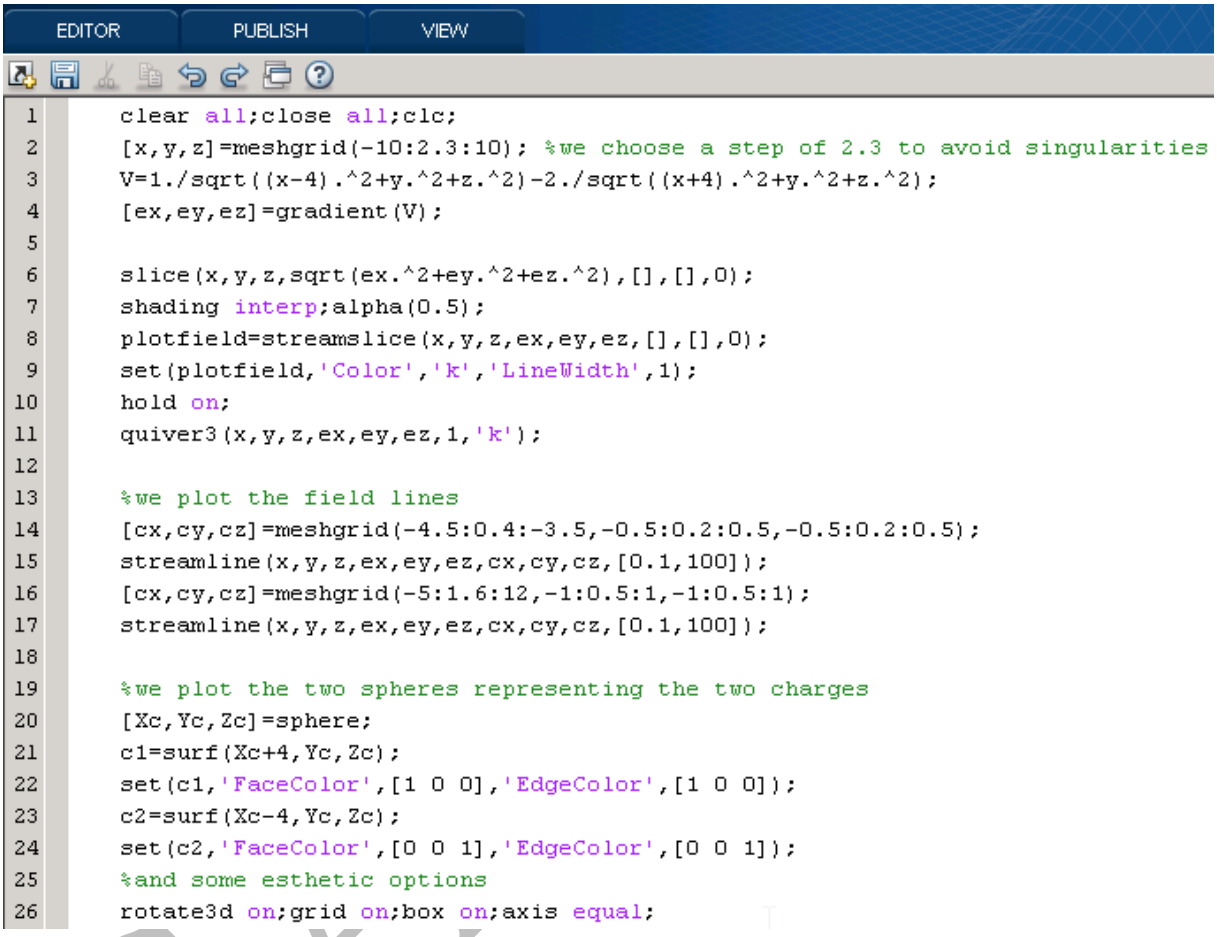
FEM Building 3D regular square grid

… **ToDo**

Vector filed equation analysis (electric field)

This is a very academic example of an electric dipole a MATLAB. In the facts it is very easy to get such an analysis with Maple but with MATLAB this is very a pain in the a... because we have to manager singularities and play with the meshgrid steps to get a relevant result.

Thus consider the following example:



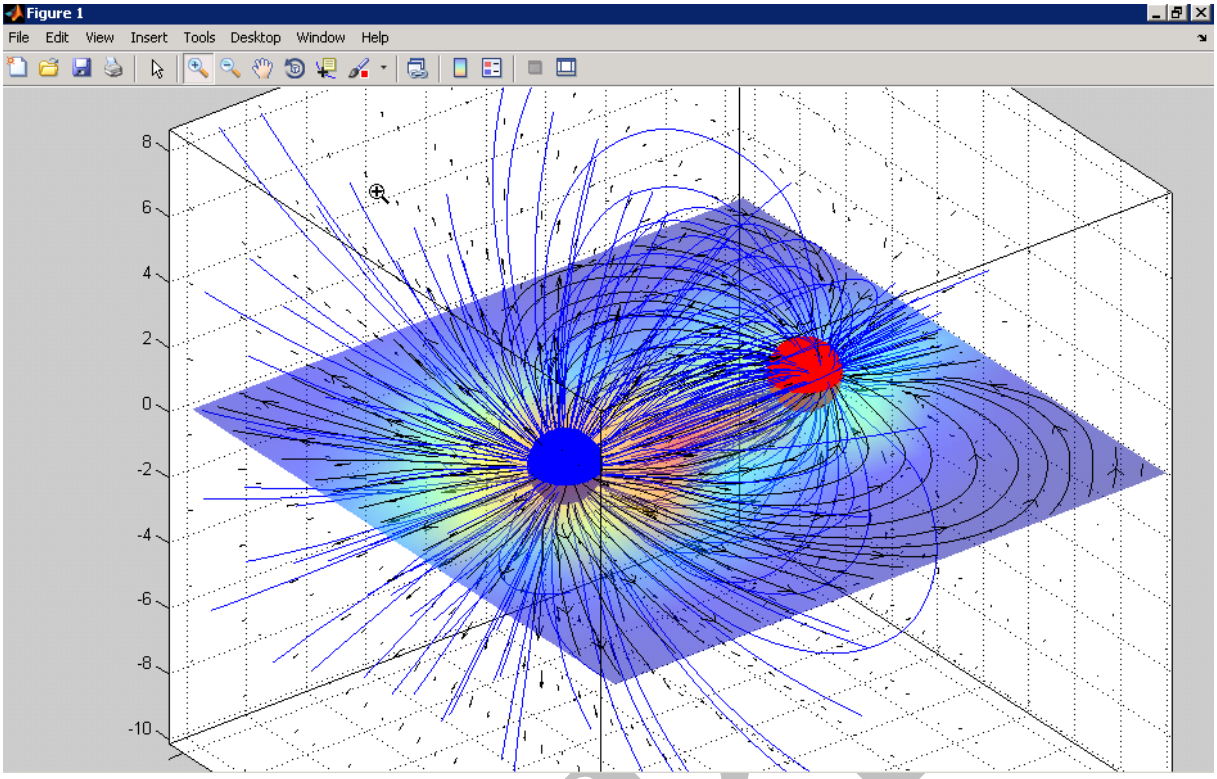

```
clear all;close all;clc;
[x,y,z]=meshgrid(-10:2.3:10); %we choose a step of 2.3 to avoid singularities
V=1./sqrt((x-4).^2+y.^2+z.^2)-2./sqrt((x+4).^2+y.^2+z.^2);
[ex,ey,ez]=gradient(V);

slice(x,y,z,sqrt(ex.^2+ey.^2+ez.^2),[],[],0);
shading interp;alpha(0.5);
plotfield=streamslice(x,y,z,ex,ey,ez,[],[],0);
set(plotfield,'Color','k','LineWidth',1);
hold on;
quiver3(x,y,z,ex,ey,ez,1,'k');

%we plot the field lines
[cx,cy,cz]=meshgrid(-4.5:0.4:-3.5,-0.5:0.2:0.5,-0.5:0.2:0.5);
streamline(x,y,z,ex,ey,ez,cx,cy,cz,[0.1,100]);
[cx,cy,cz]=meshgrid(-5:1.6:12,-1:0.5:1,-1:0.5:1);
streamline(x,y,z,ex,ey,ez,cx,cy,cz,[0.1,100]);

%we plot the two spheres representing the two charges
[Xc,Yc,Zc]=sphere;
c1=surf(Xc+4,Yc,Zc);
set(c1,'FaceColor',[1 0 0],'EdgeColor',[1 0 0]);
c2=surf(Xc-4,Yc,Zc);
set(c2,'FaceColor',[0 0 1],'EdgeColor',[0 0 1]);
%and some esthetic options
rotate3d on;grid on;box on;axis equal;
```

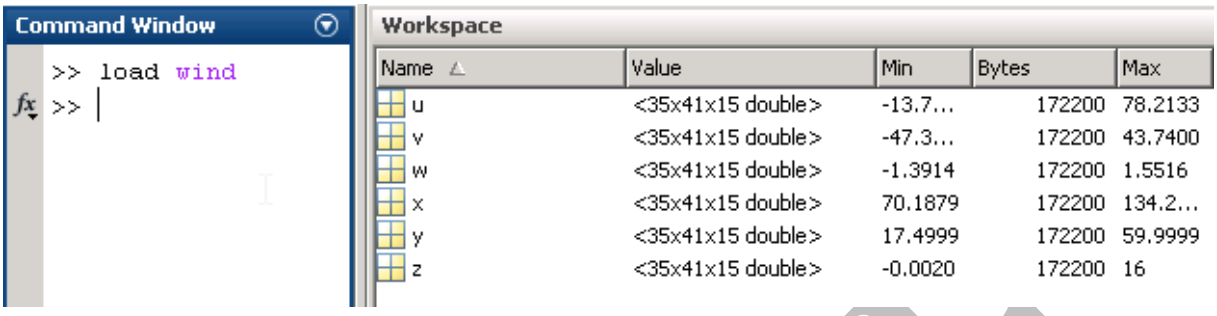
The result will be after a small zoom (see on next page):


Figure 1
File Edit View Insert Tools Desktop Window Help
8
6
4
2
0
-2
-4
-6
-8
-10

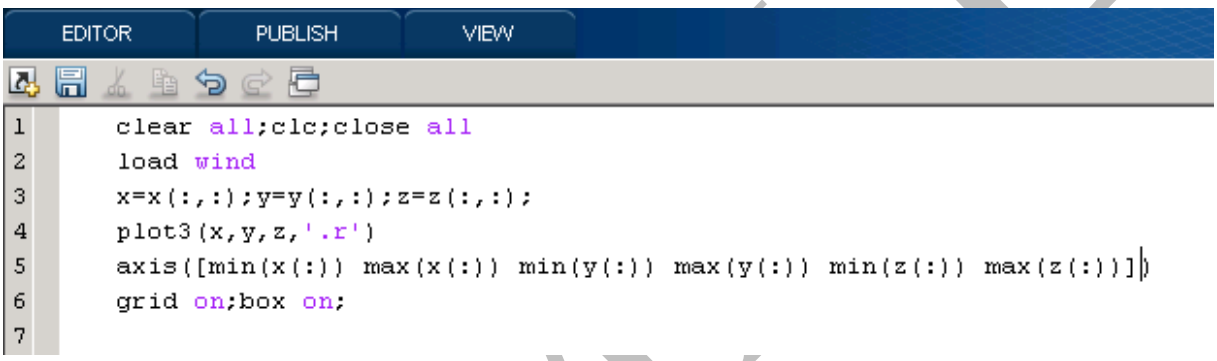
SAMPLE

Vector field snapshots analysis

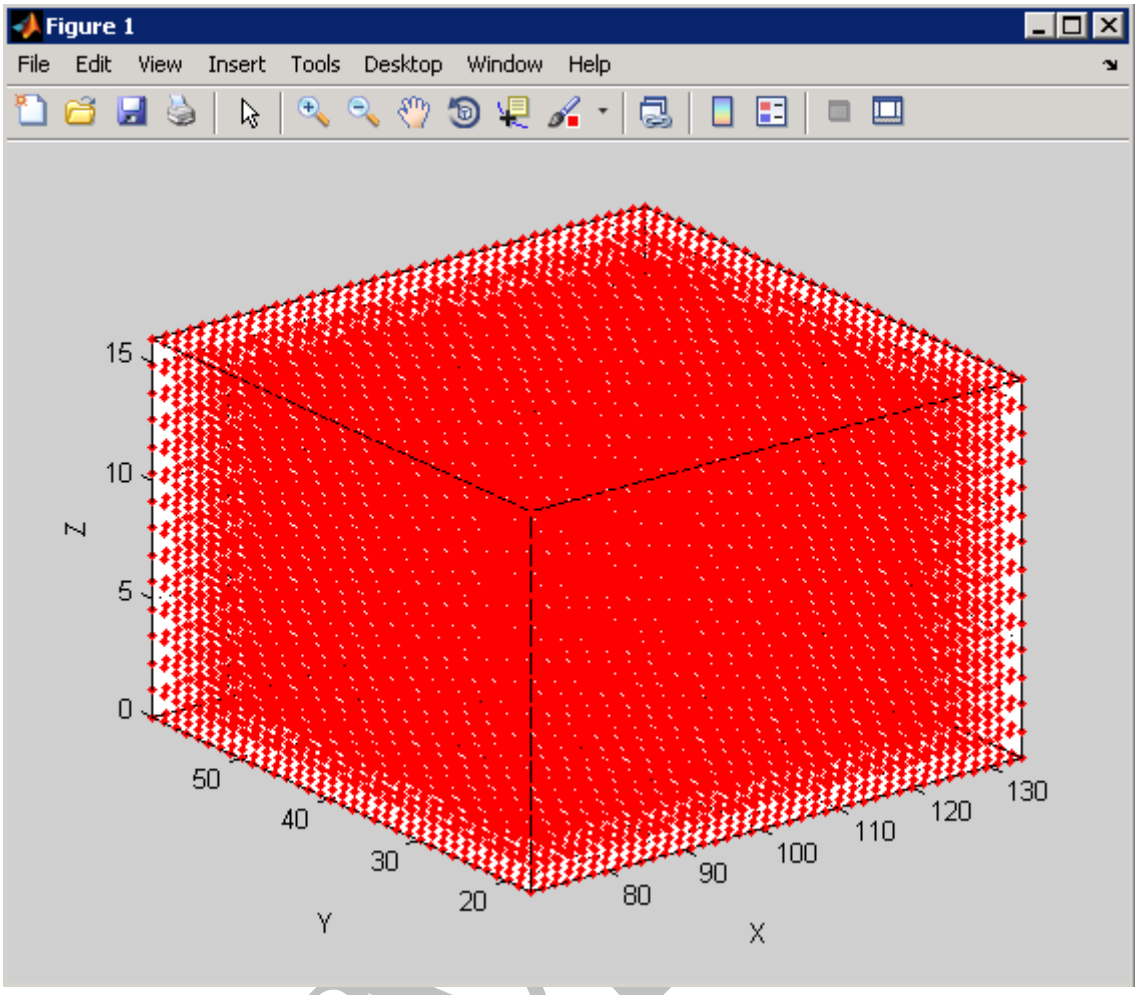
You have to imagine that we have measured (or calculated) in a spatial constant (*x*, *y*, *z*) grid (of size $35 \times 41 \times 15$) the velocity of the wind.



We can see here below the grid measure box:



SAMPLE



Now we take a look closer to velocities following the notation seen in fluid mechanics theory:

$$\left(v_x, v_y, v_z\right) = \left(u, v, w\right)$$

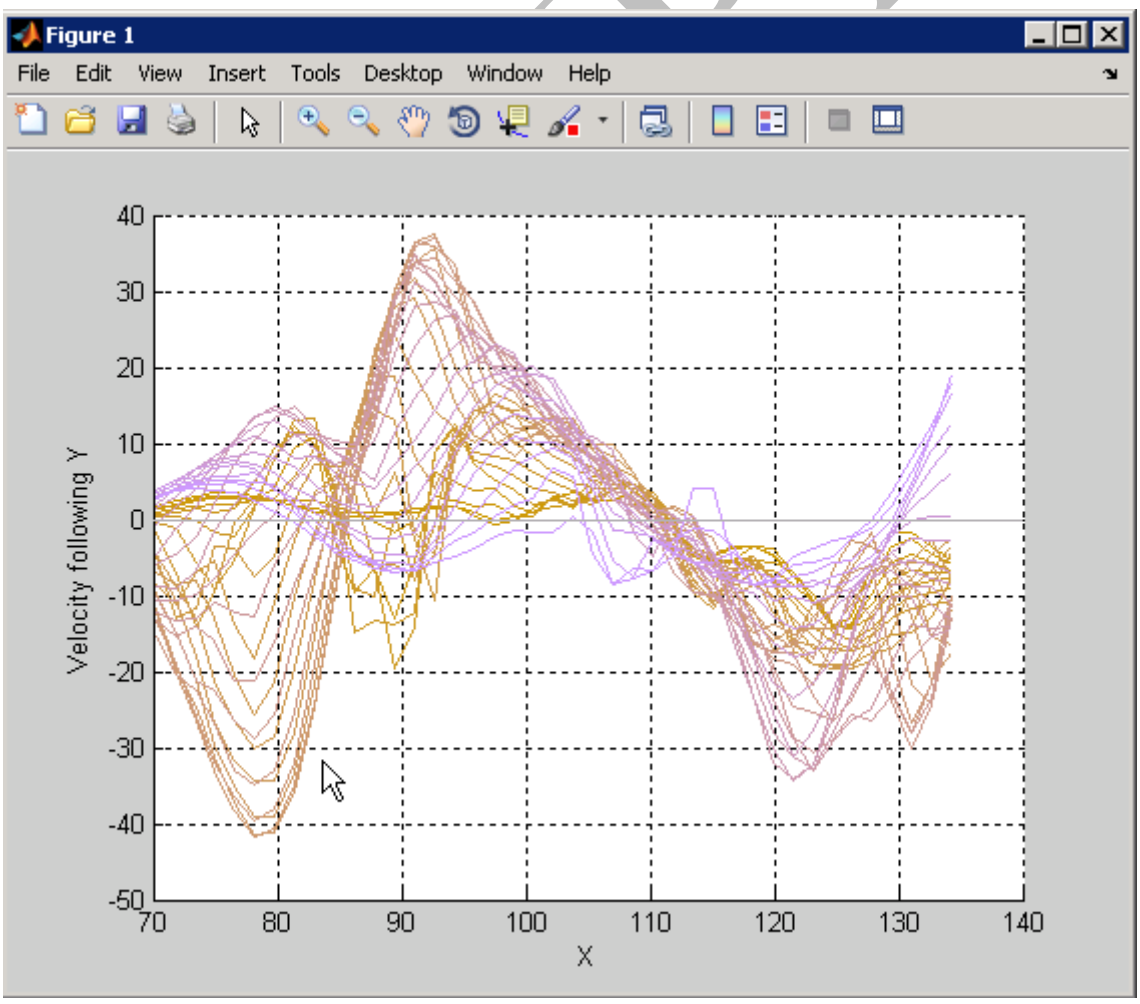
Because in our situation we have a grid of $35 \times 41 \times 15$ we have also so the same quantity of $\left(v_x, v_y, v_z\right)$ measures:

u	<35x41x15 double>
v	<35x41x15 double>
w	<35x41x15 double>
x	<35x41x15 double>
y	<35x41x15 double>
z	<35x41x15 double>

We can now take a special look at the velocity following Y:

```
clear all;clc;close all
%by curiosity we take a look of the velocity following Y, for a given Z
%measure line and multiples X
Z=7; %This fixed the Z measure line height (empirical choice)
load wind
for coord=1:1:35
    %remember that "v" is the velocity following "Y"
    v_plot=v(coord,:,Z);
    x_plot=x(coord,:,Z);
    hold on;
    grid on;
    plot(x_plot,v_plot,'Color',[0.8 0.6 coord/35])
end
%the zero line
hy = graph2d.constantline(0, 'Color',[.7 .7 .7]);
changedependvar(hy,'y');
xlabel('X');ylabel('Velocity following Y');
```

This will give:

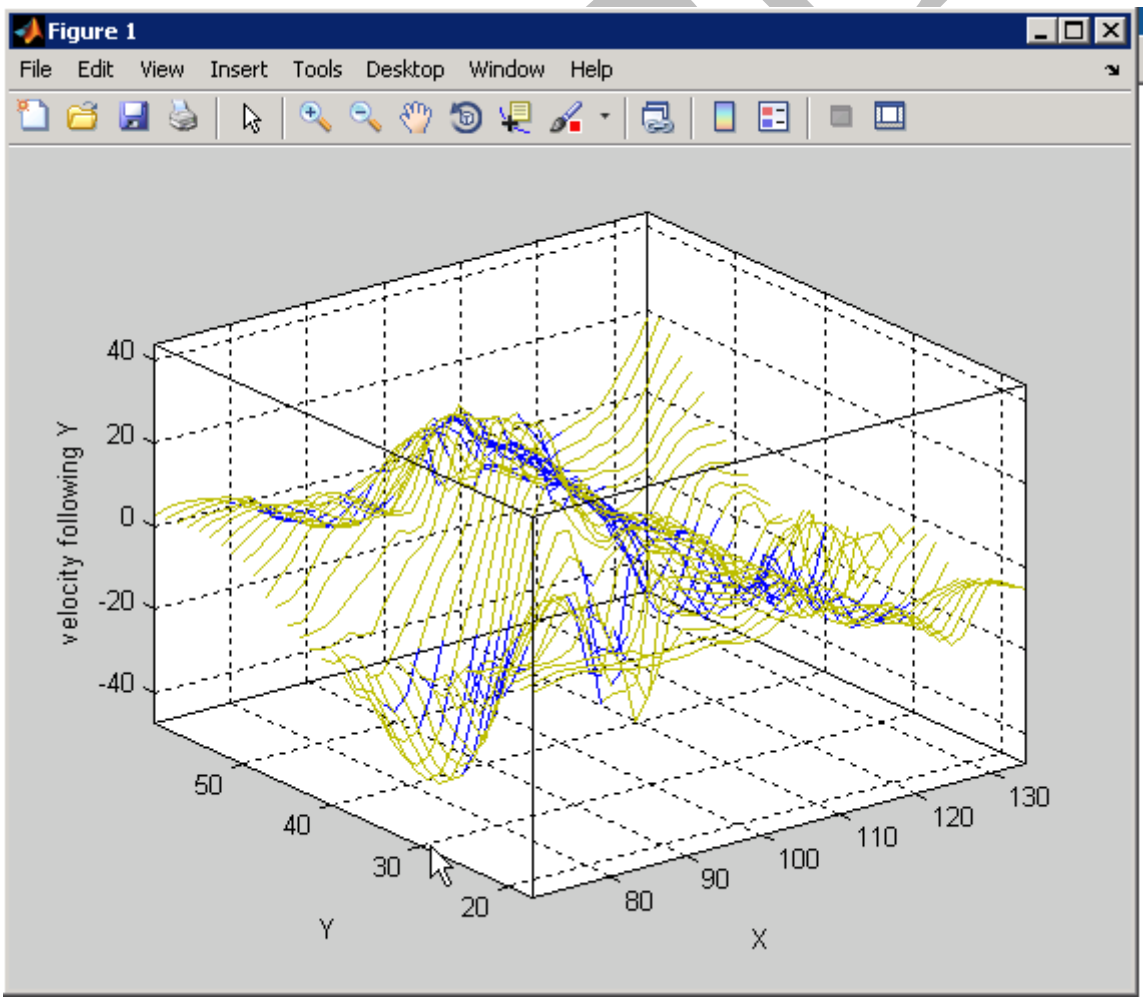


We can see above a speed inversion when *X* is between [70,80].

Or in 3D:

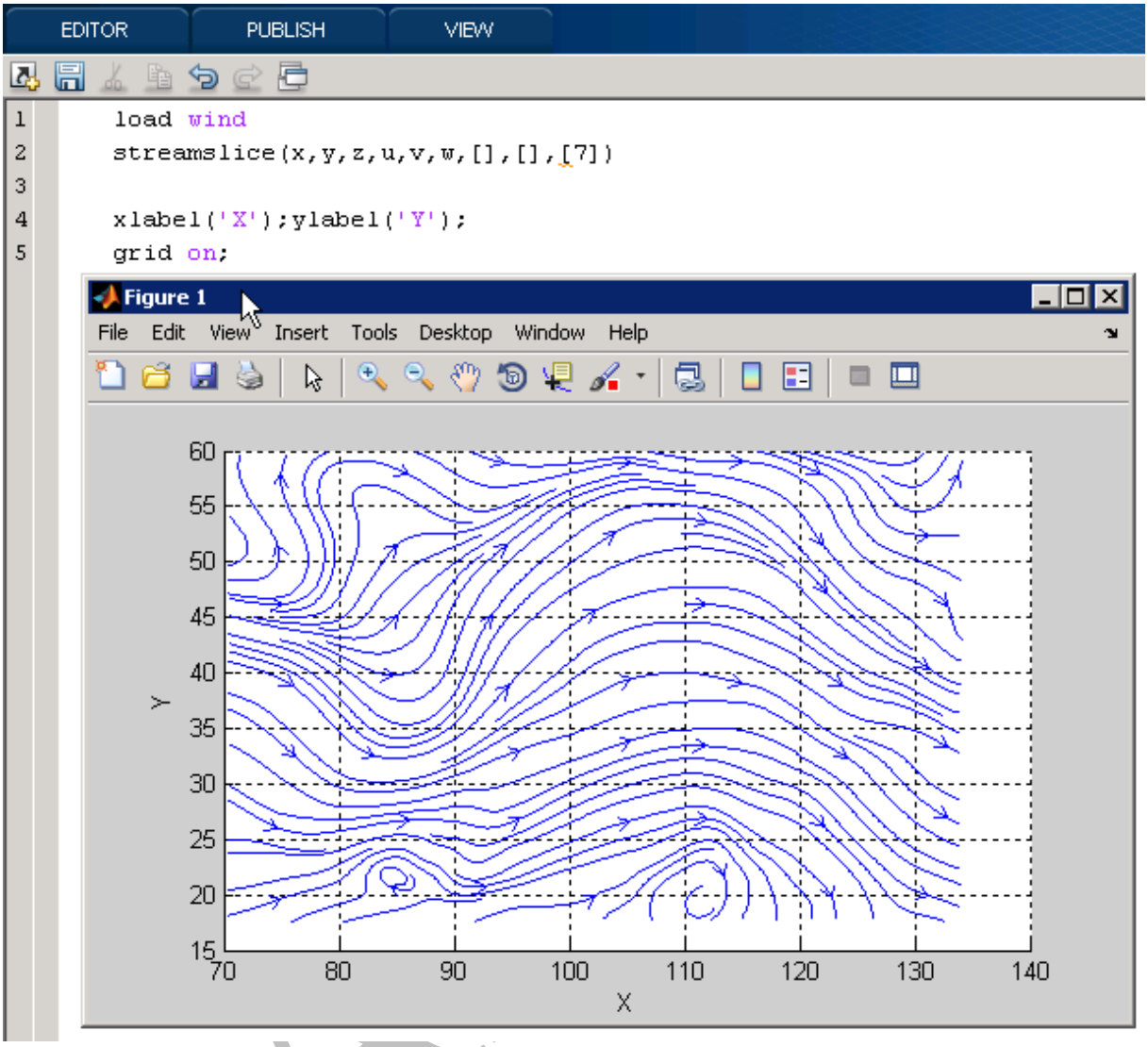
```
clear all;clc;close all;
load wind;
Z=7;
for coord=1:1:35
    plot3(x(coord,:,1),repmat(y(coord,1,Z),41),v(coord,:,Z));
    hold on;
end
axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(v(:)) max(v(:))])
grid on;box on;
xlabel('X');ylabel('Y');zlabel('velocity following Y');
```

This will give:



ToDo: make the same faster with waterfall

We can se a profile for the chosen Z using the function **streamslice( )**:

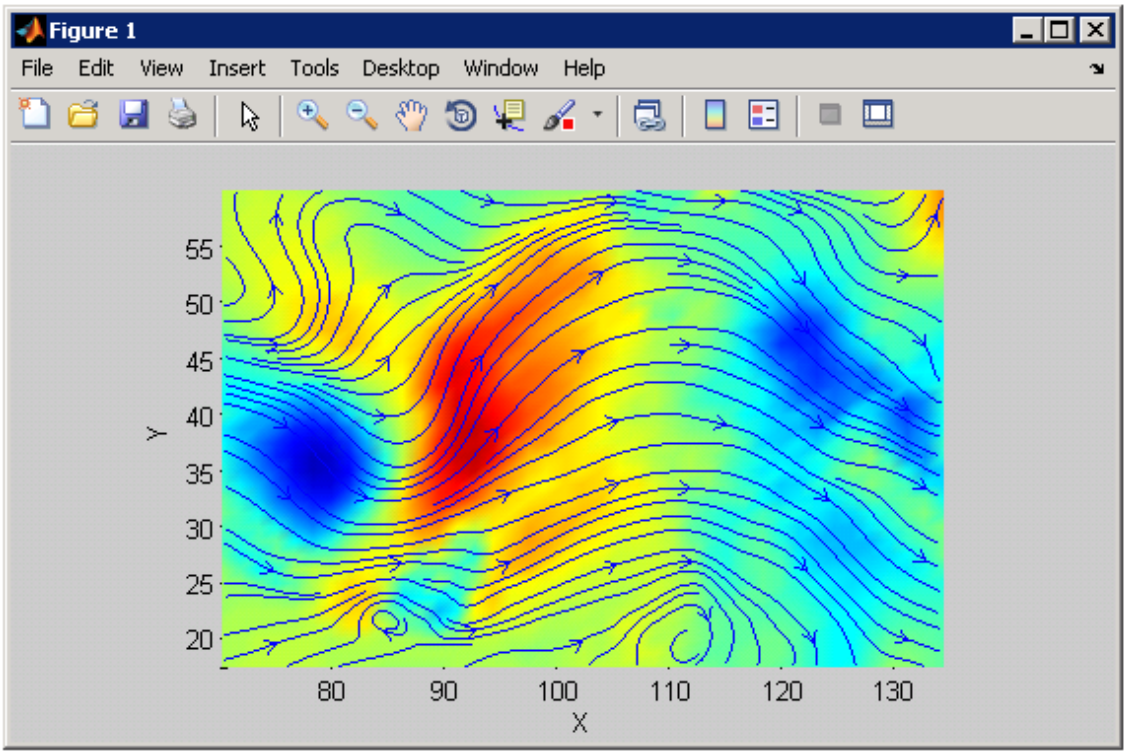


Thus we understand better the inversion of the velocity following *X* in the range [70,90].

Or in another way:

EDITOR PUBLISH VIEW

```
1    load wind
2    slice(x,y,z,v,[],[],[7])
3    streamslice(x,y,z,u,v,w,[],[],[7])
4    axis equal
5    shading interp
6    xlabel('X');ylabel('Y');
7    view(2)
```

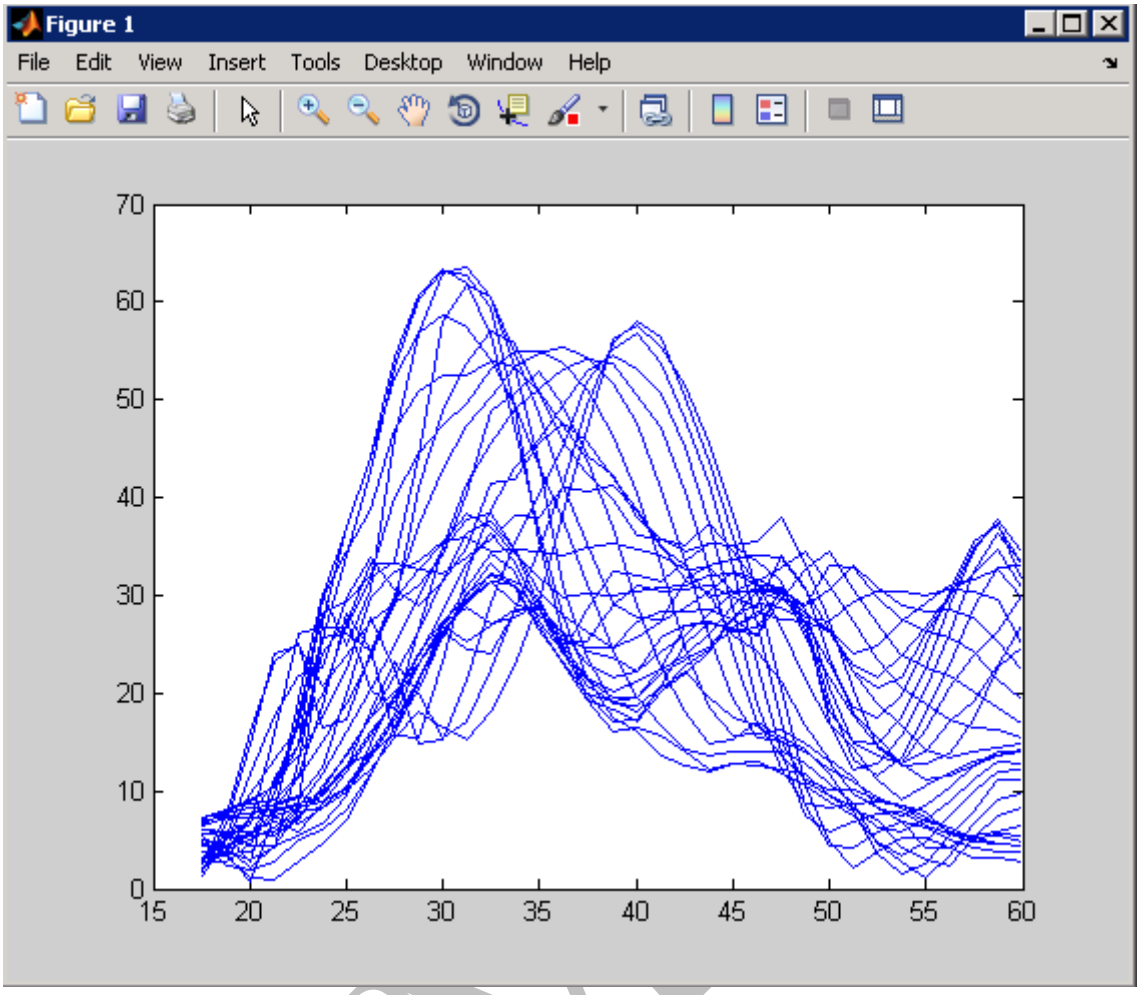


That helps to understand the amplitude of velocity!

We can also see the same thing by taking the norm of the velocity in the (*x*, *y*) plane for the same *Z* for each *Y*:

EDITOR PUBLISH VIEW

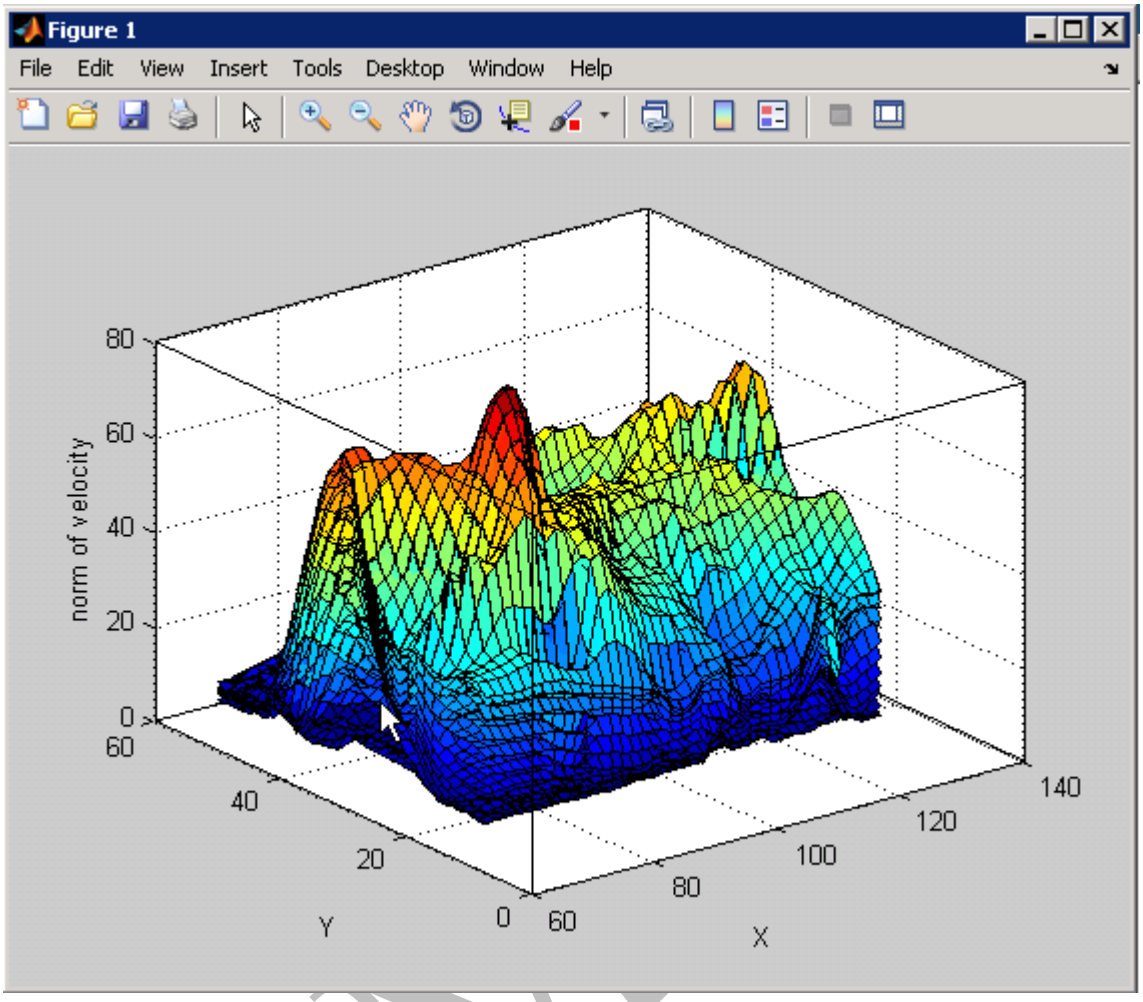
```
1    clear all;clc;close all
2    Z=7;
3    for coord=1:1:31
4        load wind
5        u=u(:,coord,Z);v=v(:,coord,Z);y=y(:,coord,Z);
6        spd = sqrt(u.*u + v.*v);
7        plot(y,spd)
8        hold on
9    end
```



We can also take a close look to the norm of the velocity for different *Z* and all (*x*, *y*):

EDITOR PUBLISH VIEW

```
1   clear all;clc;close all
2   for k=1:15
3       for coord=1:31
4           load wind
5           u=u(:,:,k);v=v(:,:,k);w=w(:,:,k);
6           x=x(coord,:,k);y=y(:,coord,k);
7           spd = sqrt(u.*u + v.*v+w.*w);
8           surf(x,y,spd)
9           hold on
10      end
11  end
12  grid on;box on;
13  xlabel('X');ylabel('Y');zlabel('norm of velocity');
```

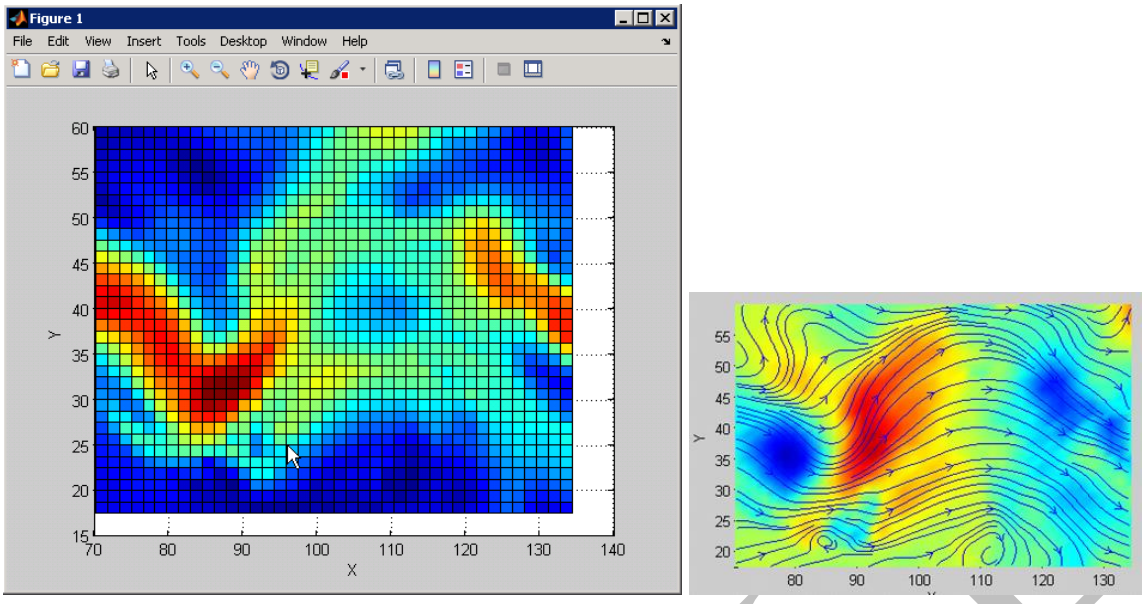


As we can see: deeper we are, slower is the velocity. In the borders velocity is also low. This corresponds to the streamslice!

But if we focus only on the 7th level of *Z* we see that:

EDITOR PUBLISH VIEW

```
1   clear all;clc;close all
2   for k=7
3       for coord=1:31
4           load wind
5           u=u(:,:,k);v=v(:,:,k);w=w(:,:,k);
6           x=x(coord,:,k);y=y(:,coord,k);
7           spd = sqrt(u.*u + v.*v+w.*w);
8           surf(x,y,spd)
9           hold on
10      end
11  end
12  grid on;box on;
13  xlabel('X');ylabel('Y');zlabel('norm of velocity');
14  view(2)
```



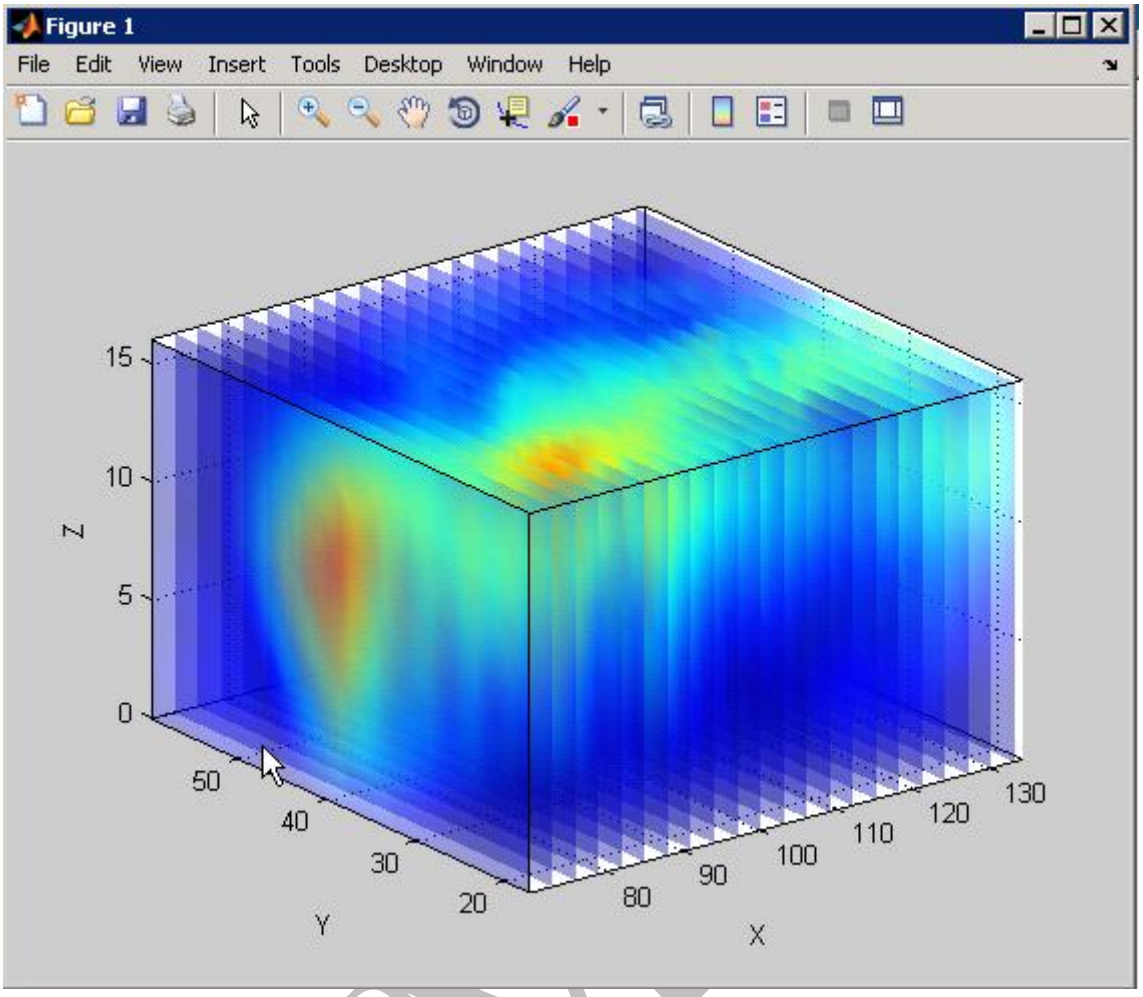
Then we see that the streamslice take (for sure...) in comparison the global amplitude for all *Z*!

In a more advanced way we can see the norm of the velocity in 3D with interpolation using the function **slice( )**:

EDITOR PUBLISH VIEW

```
1     clear all;clc;close all
2     load wind;
3     xmin=min(x(:));xmax=max(x(:));
4     wind_speed=sqrt(u.^2+v.^2+w.^2);
5     hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[],[]);
6     set(hsurfaces,'FaceColor','interp','EdgeColor','none');
7     alpha(0.4)
8     grid on;box on;
9     xlabel('X');ylabel('Y');zlabel('Z');
10    axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```

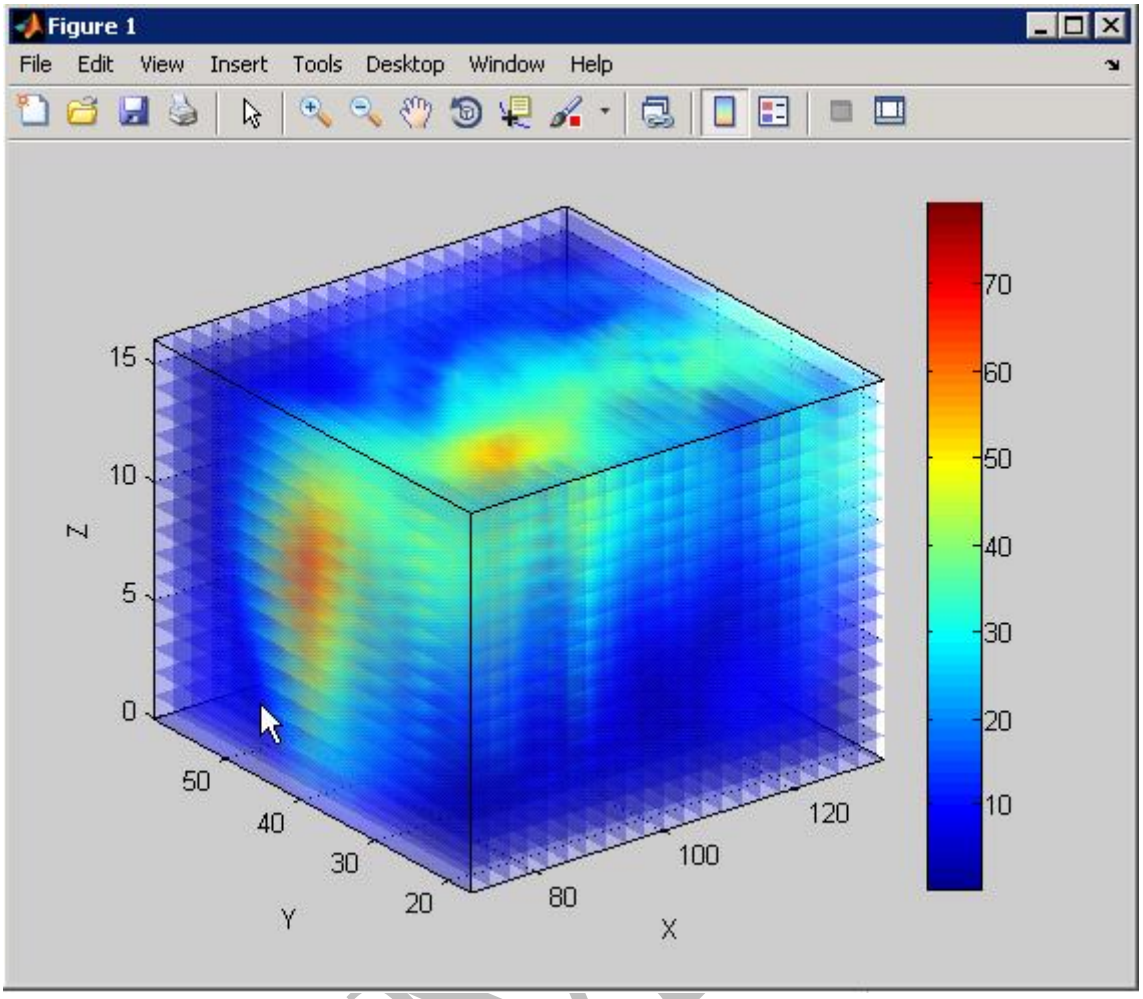
We can see an expected result: the velocity is low in the borders of the box a high velocities are concentrated in the center.



Or if you prefer something more "dense":

EDITOR PUBLISH VIEW

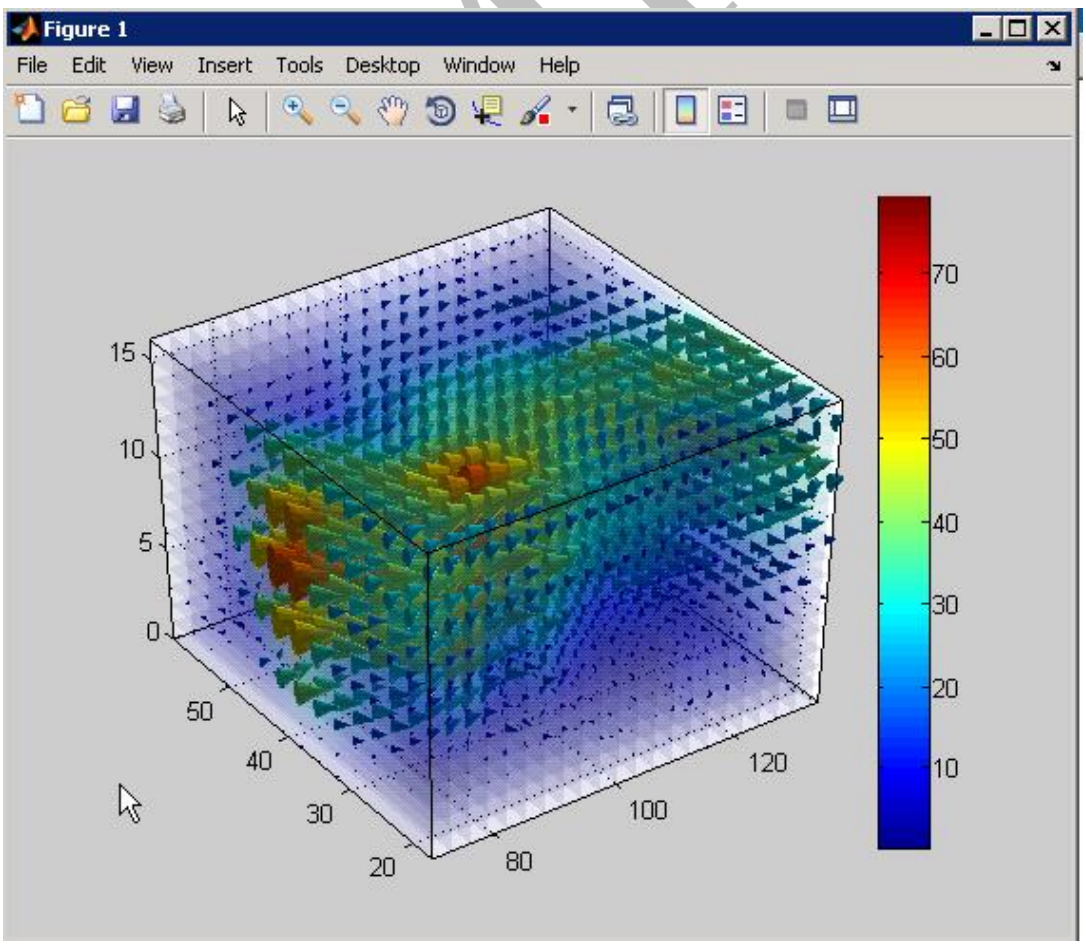
```
1   clear all;clc;close all
2   load wind;
3   zmin=min(z(:));zmax=max(z(:));
4   xmin=min(x(:));xmax=max(x(:));
5   wind_speed=sqrt(u.^2+v.^2+w.^2);
6   hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[],[zmin:1:zmax]);
7   set(hsurfaces,'FaceColor','interp','EdgeColor','none');
8   alpha(0.3);
9   colorbar;
10  grid on;box on;
11  xlabel('X');ylabel('Y');zlabel('Z');
12  axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```


Figure 1
File Edit View Insert Tools Desktop Window Help
15
10
5
0
Z
50
40
30
20
Y
80
100
120
X
70
60
50
40
30
20
10

SAM

Or in a different way using the function **coneplot( )**:

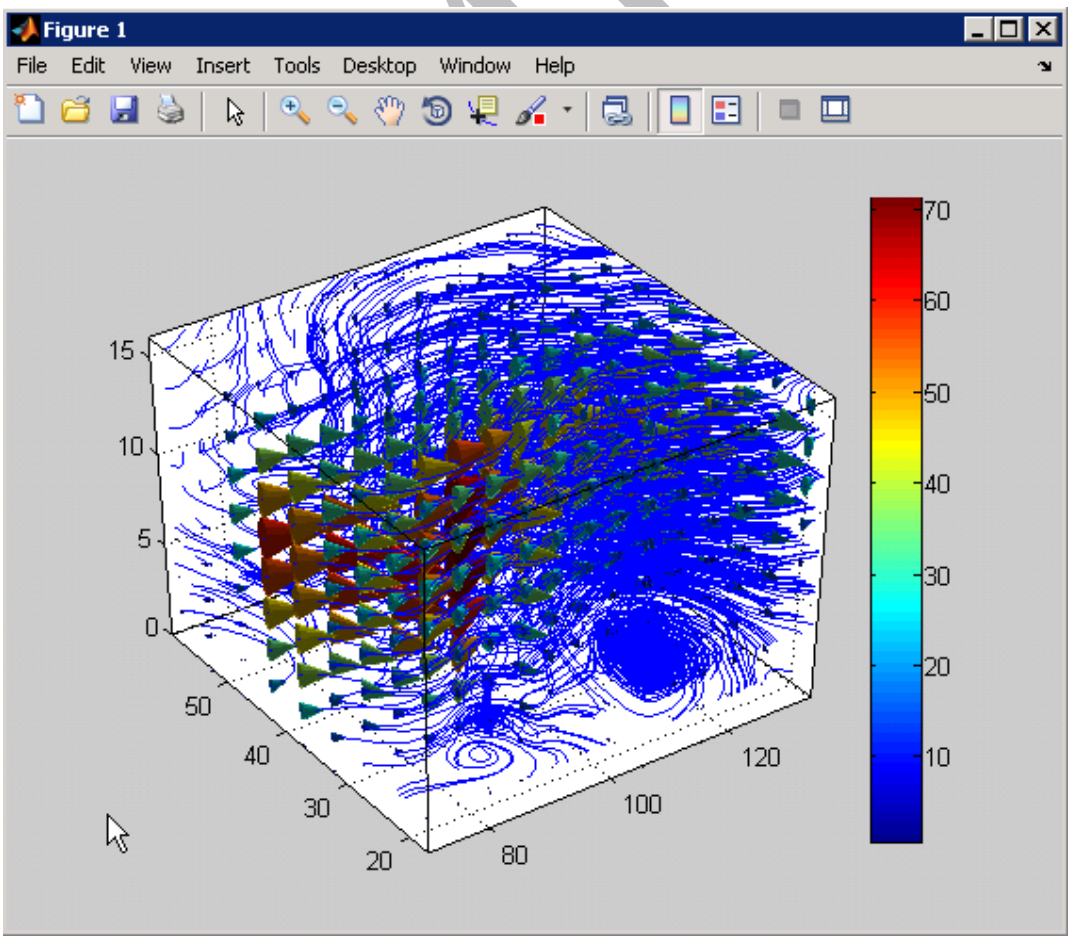
```
clear all;clc;close all
load wind;
xmin = min(x(:));xmax = max(x(:));ymin = min(y(:));ymax = max(y(:));
zmin = min(z(:));zmax = max(z(:));
wind_speed = sqrt(u.^2 + v.^2 + w.^2);
hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[],[zmin:1:zmax]);
set(hsurfaces,'FaceColor','interp','EdgeColor','none');
alpha(0.1);
colors = wind_speed;
scale = 4;
[cx, cy, cz] = meshgrid(xmin:3:xmax,ymin:3:ymax,zmin:3:zmax);
c = coneplot(x,y,z,u,v,w,cx,cy,cz,scale,colors);
set(c,'EdgeColor','none');
view([-35 40]);
camproj perspective;
grid on;box on;
light;lighting flat;
colorbar;
xlabel('X');ylabel('Y');zlabel('Z');
axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```



Or something a little bit more interesting using the function **streamline( )**:

EDITOR PUBLISH VIEW

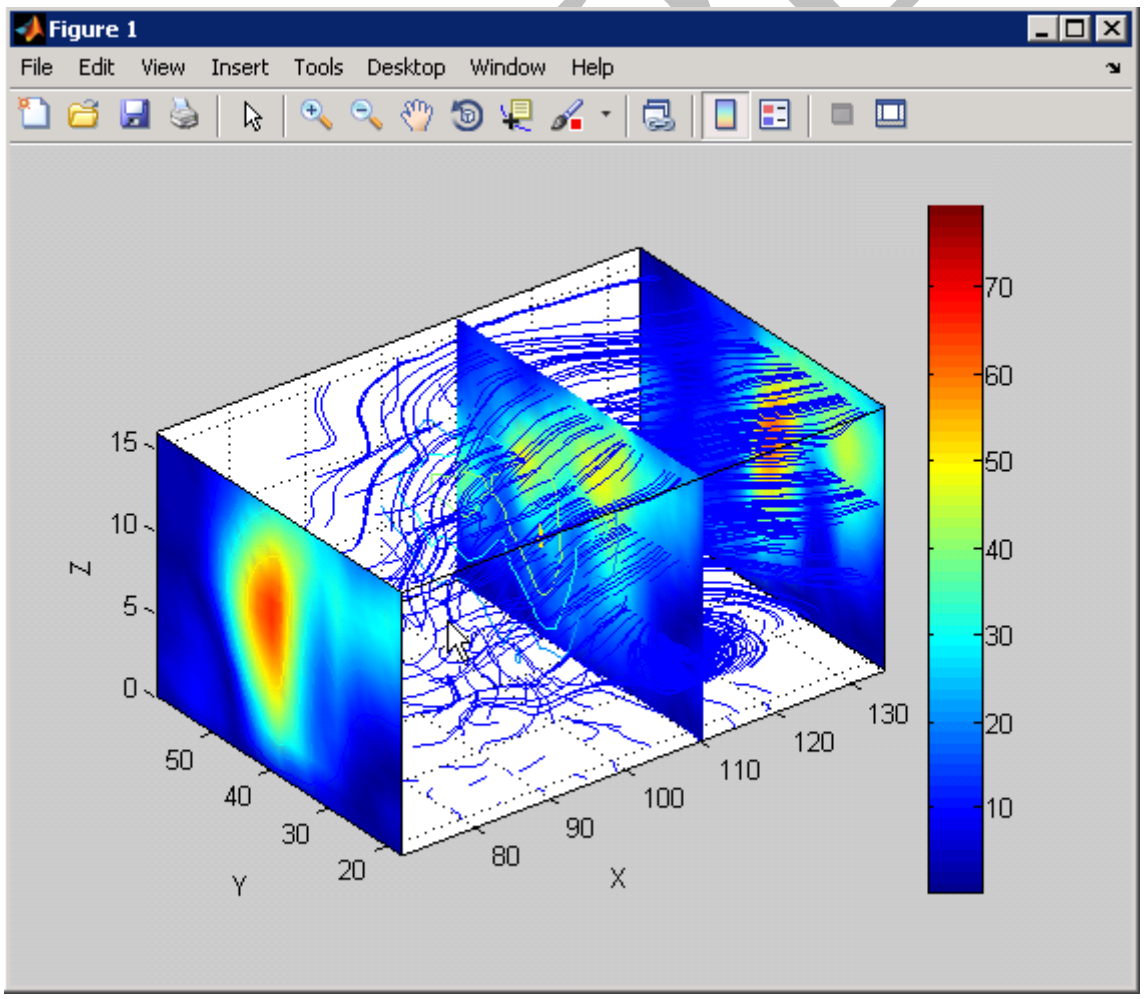
```
clear all;clc;close all
load wind;
xmin = min(x(:));xmax = max(x(:));ymin = min(y(:));ymax = max(y(:));
zmin = min(z(:));zmax = max(z(:));
wind_speed = sqrt(u.^2 + v.^2 + w.^2);
colorbar('vertical');
colors = wind_speed;
scale=4;
[cx, cy, cz] = meshgrid(xmin:5:xmax,ymin:5:ymax,zmin:2:zmax);
c = coneplot(x,y,z,u,v,w,cx,cy,cz,scale,colors);
set(c,'EdgeColor','none');
h = streamline(x,y,z,u,v,w,cx,cy,cz);
view([-35 40]);
camproj perspective;
grid on;box on;
light;lighting flat;
colorbar;
xlabel('X');ylabel('Y');zlabel('Z');
axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```



Or with another style using the function **contourslice( )**:

EDITOR PUBLISH VIEW

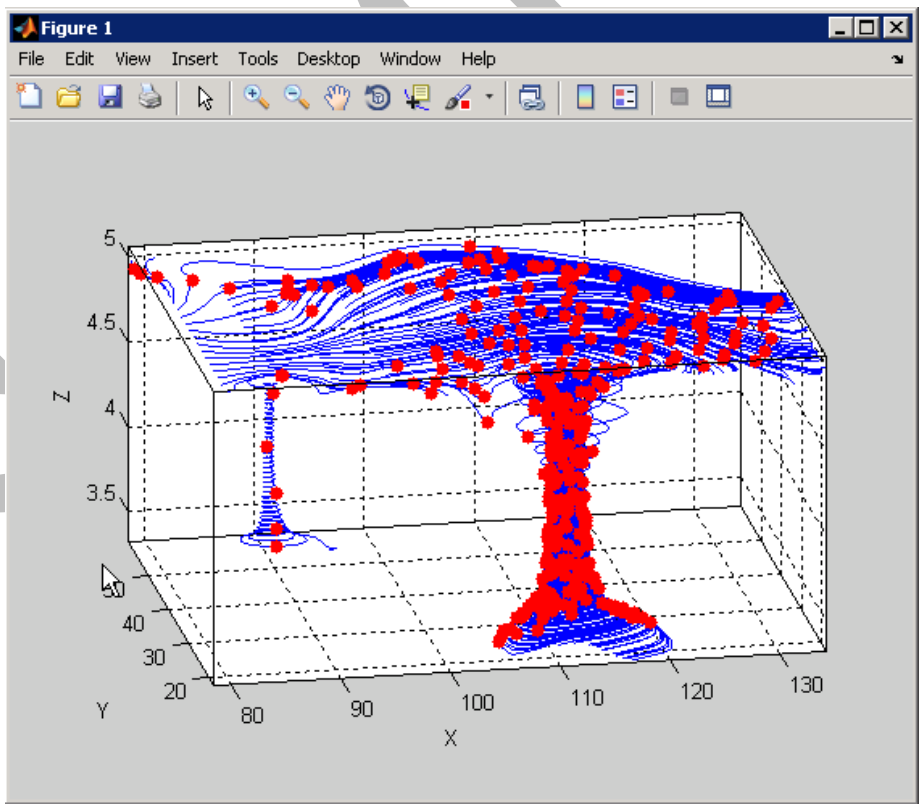
```
clear all;clc;close all
load wind;
xmin=min(x(:));xmax=max(x(:));ymax=max(y(:));zmin=min(z(:));
wind_speed=sqrt(u.^2+v.^2+w.^2);
hsurfaces=slice(x,y,z,wind_speed,[xmin,110,xmax],[],[]);
set(hsurfaces,'FaceColor','interp','EdgeColor','none');
hcont=contourslice(x,y,z,wind_speed,[xmin,100,xmax],[],[]);
[sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
hlines=streamline(x,y,z,u,v,w,sx,sy,sz);
set(hlines,'LineWidth',1,'Color','b');
xlabel('X');ylabel('Y');zlabel('Z');
view(3); daspect([2,2,1]); axis tight;
grid on;box on;
colorbar;
axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```



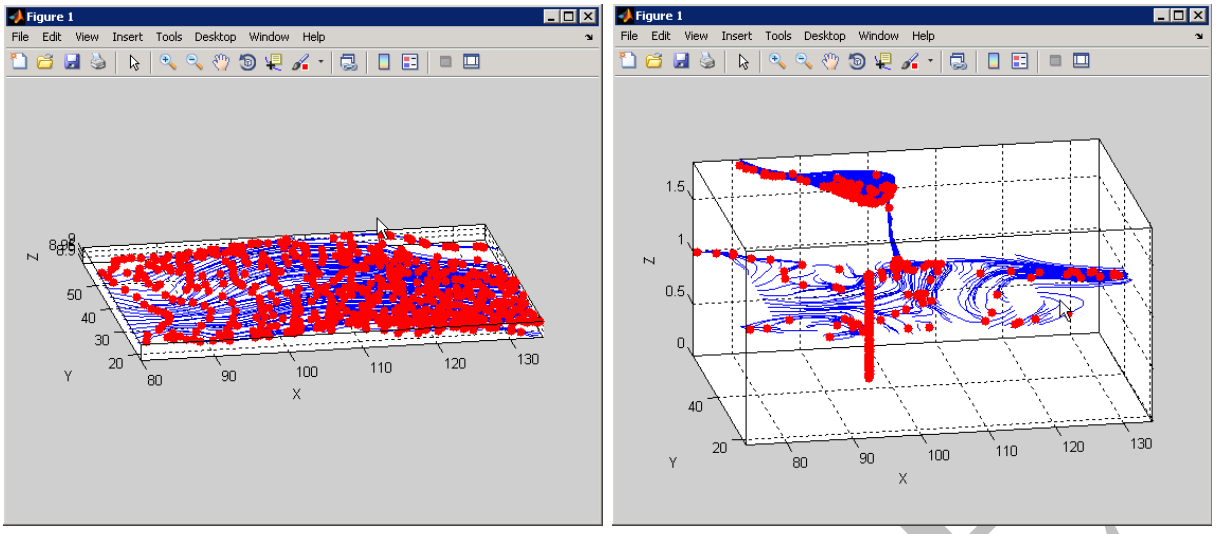
We can also animate that stuff starting from a given height using the function **interpstreamspeed( )** and **streamparticles( )**:

```
clear all;clc;close all
step=5;
Zheight=5;
load wind;
[sx,sy,sz]=meshgrid(80:step:120,20:2:50,Zheight);
verts=stream3(x,y,z,u,v,w,sx,sy,sz);
sl=streamline(verts);
view(-10.5,18);
daspect([2 2 0.125]);
axis tight;box on;grid on
iverts =interpstreamspeed(x,y,z,u,v,w,verts,0.05);
xlabel('X');ylabel('Y');zlabel('Z');
set(gca,'drawmode','fast'); streamparticles(iverts,15,'Animate',10,...
    'ParticleAlignment','on','MarkerEdgeColor','none','MarkerFaceColor',...
    'red','Marker','o');
axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);
```

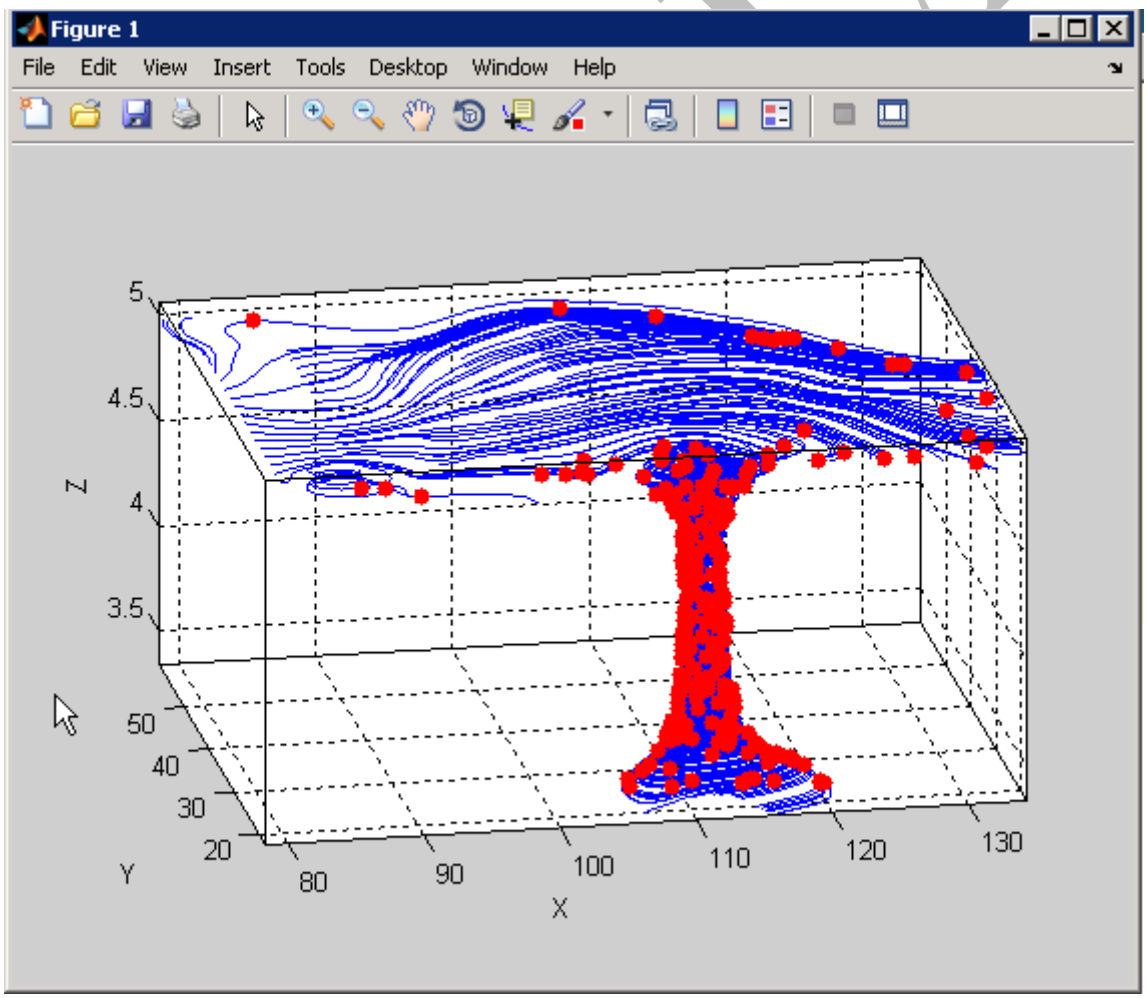
This will give:



If we change de *Zheigth* to 9 on the left figure below (or to 1 on the figure on the right below) the result is quite impressive in comparison:



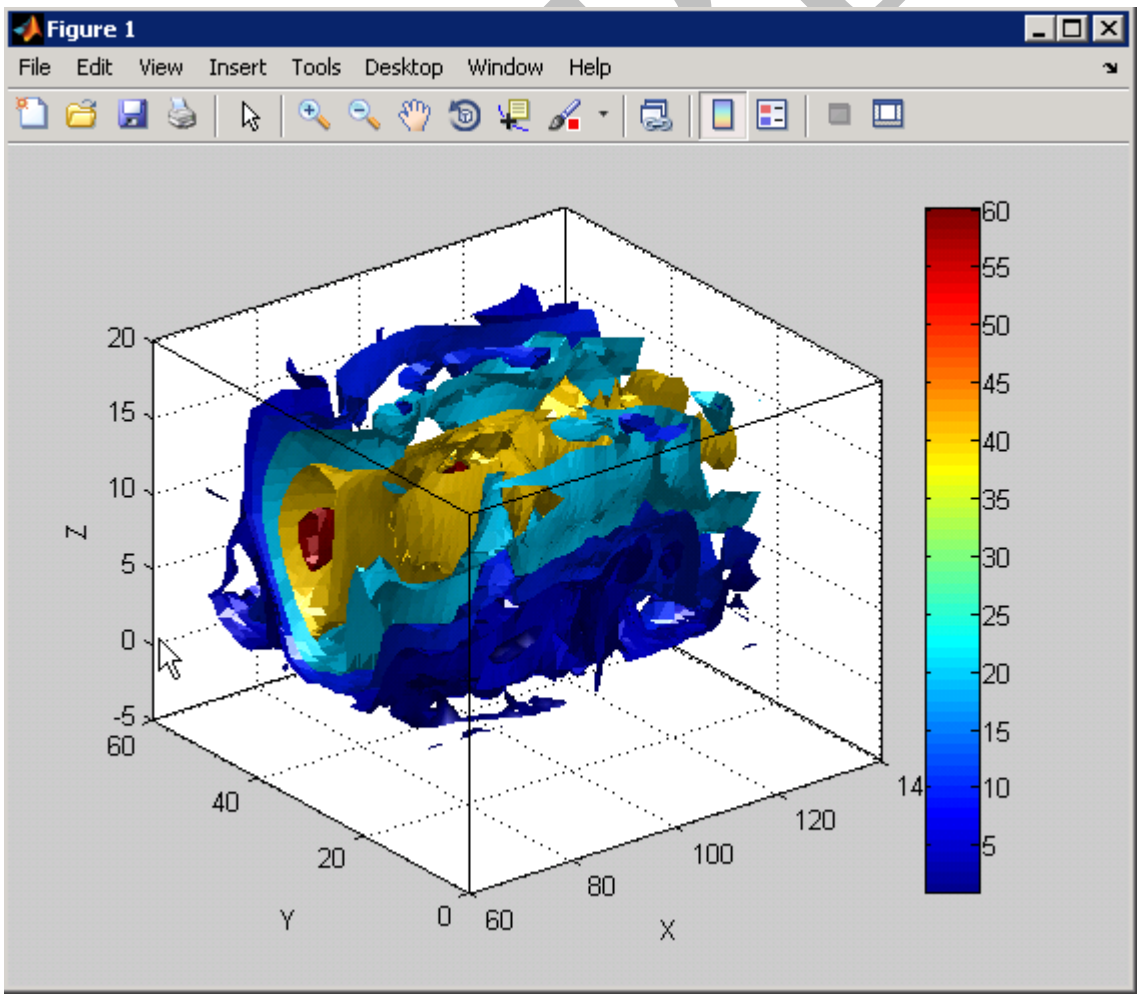
Or for the same *Zheight* (5) and with a *steps* of 6 we also have a surprise (for sure... to big step hide information):



Or we can also play with the function **isosurfaces( )**:

```
1   load wind;
2   spd = sqrt(u.*u + v.*v + w.*w);
3   % Create isosurface patch
4   isosurface(x, y, z, spd,1);
5   hold on;
6   isosurface(x, y, z, spd,5);
7   isosurface(x, y, z, spd,10);
8   isosurface(x, y, z, spd,20);
9   isosurface(x, y, z, spd,40);
10  isosurface(x, y, z, spd,60);
11  colorbar;
12  grid on;box on;
13  xlabel('X');ylabel('Y');zlabel('Z');
```

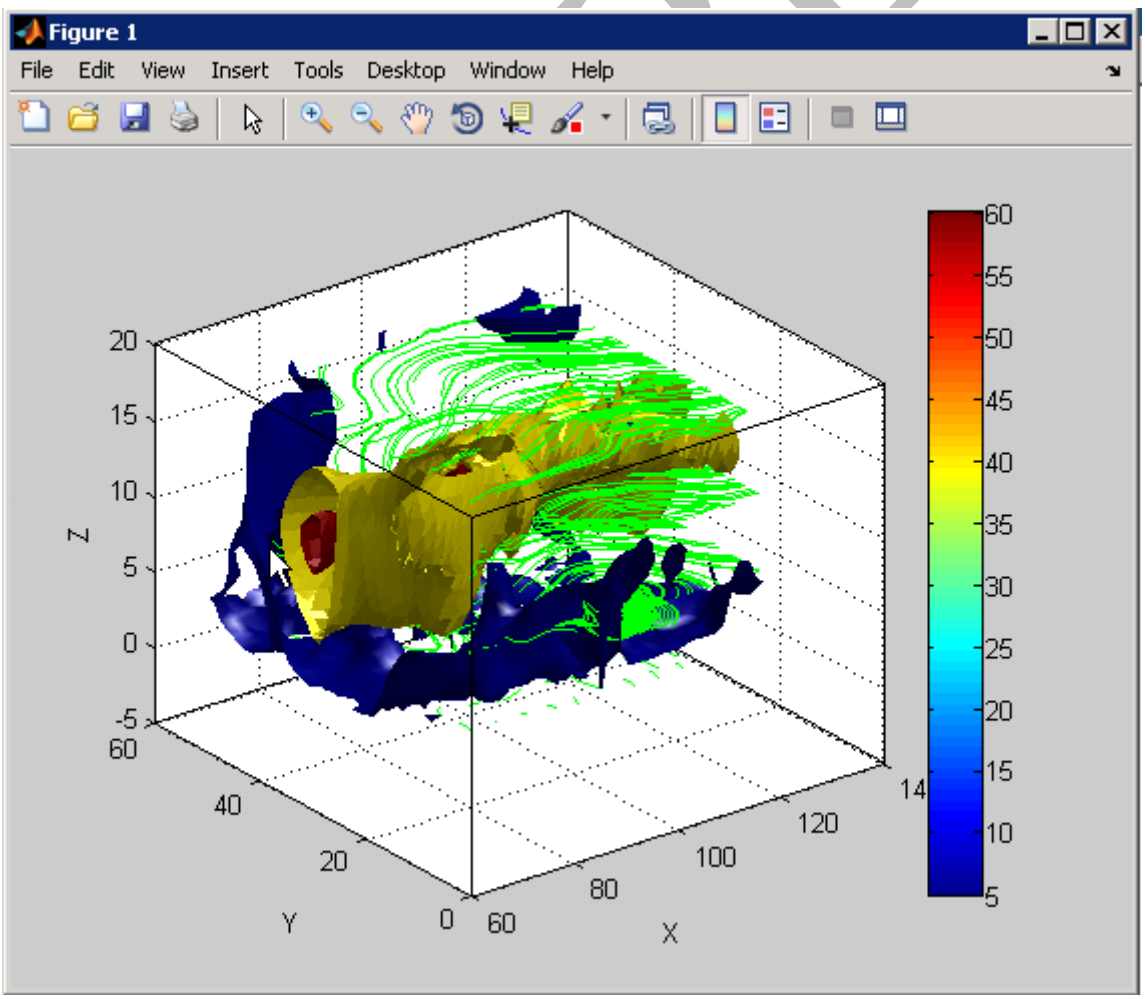
To get:



We can also add streamlines to this:

```
load wind;
spd = sqrt(u.*u + v.*v + w.*w);
% Create isosurface patch
isosurface(x, y, z, spd,5);
hold on;
isosurface(x, y, z, spd,40);
isosurface(x, y, z, spd,60);
[sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
h2 = streamline(x,y,z,u,v,w,sx,sy,sz);
set(h2,'LineWidth',1, 'Color', 'g');
colorbar;
grid on;box on;
xlabel('X');ylabel('Y');zlabel('Z');
```

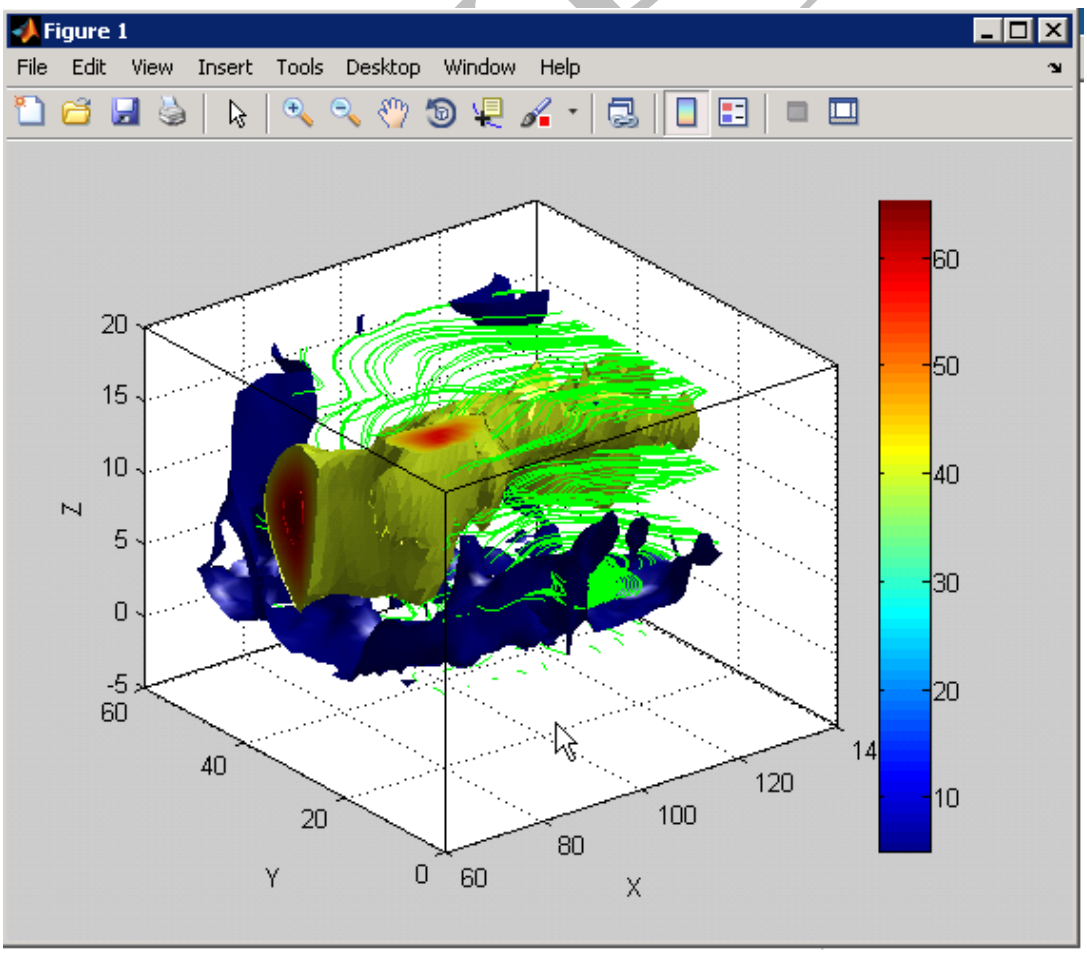
This will give:



And we can close the holes of a given isosurface using the function **isocaps( )**:

```
1     load wind;
2     spd = sqrt(u.*u + v.*v + w.*w);
3     % Create isosurface patch
4     isosurface(x, y, z, spd,5);
5     hold on;
6     isosurface(x, y, z, spd,40);
7     %we close the above isosurface
8     isocaps(x, y, z, spd, 40);
9     isosurface(x, y, z, spd,60);
10    [sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
11    h2 = streamline(x,y,z,u,v,w,sx,sy,sz);
12    set(h2,'LineWidth',1, 'Color', 'g');
13    colorbar;
14    grid on;box on;
15    xlabel('X');ylabel('Y');zlabel('Z');
```

This will give you:



etc.

Regression/Interpolation/Extropolation

OLS univariate regression

These are some simple OLS regression examples without statistical reports (because you do not need the Statistical Toolbox) using the **polyfit( )** function for the regression itself and after a simple **plot( )** for the chart.

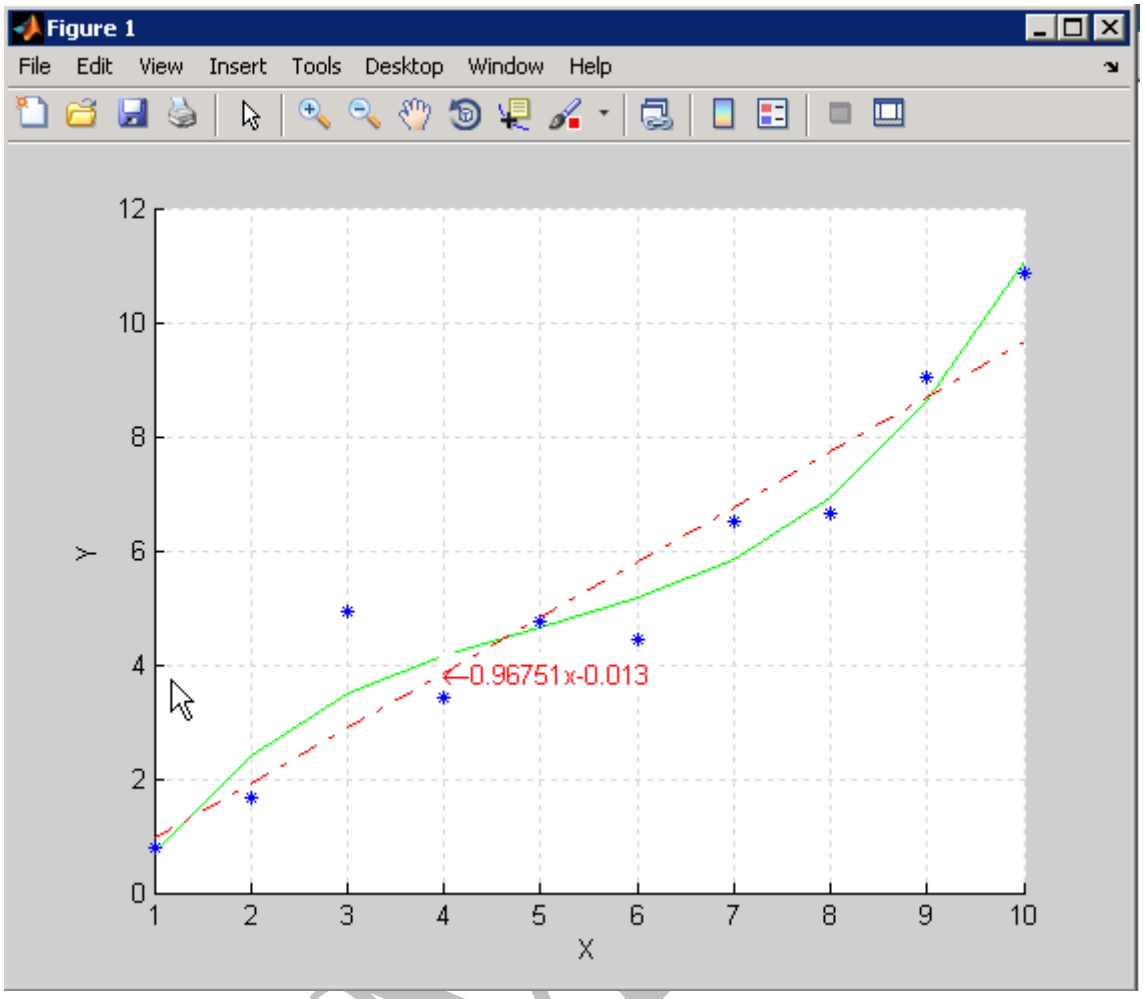
EDITOR PUBLISH VIEW

```
%First a simple OLS regression
x = 1:10;
y = x + randn(1,10);
scatter(x,y,25,'b','*')
P_line = polyfit(x,y,1);
yfit = P_line(1)*x+P_line(2);
hold on
plot(x,yfit,'r-.');
xlabel('X');ylabel('Y');
grid on;
set(gca,'Xcolor',[0.85 0.85 0.85])
set(gca,'Ycolor',[0.85 0.85 0.85])
c_axes = copyobj(gca,gcf);
set(c_axes,'color','none','xcolor','k','xgrid','off',...
    'ycolor','k','ygrid','off');

%And now the same with a third order polynom
P_poly = polyfit(x,y,3);
yfit = P_poly(1)*x.^3+P_poly(2)*x.^2+P_poly(3)*x+P_poly(4);
plot(x,yfit,'-g');
P_poly(1);P_poly(2);P_poly(3);

%we add some texts (att the end to be overs the other plots)
text(4,polyval(P_line,4),...
    strcat('\leftarrow',num2str(P_line(1)),'x',...
    sprintf('%+3.3f',P_line(2))),'Color','r','BackgroundColor',[1 1 1]);
```

The result is visible on the next page:


Figure 1
File Edit View Insert Tools Desktop Window Help
12
10
8
6
4
2
0
Y
1 2 3 4 5 6 7 8 9 10
X
←0.96751x-0.013

SAMPLE

Spline univariate interpolation

Don't forget that with spline interpolation the main definition support must be monotonic increasing and extrapolate before or after the first/last point is a non-sense (this is also why you can't use it finance to extrapolate the first and last missing values of a return vector!):

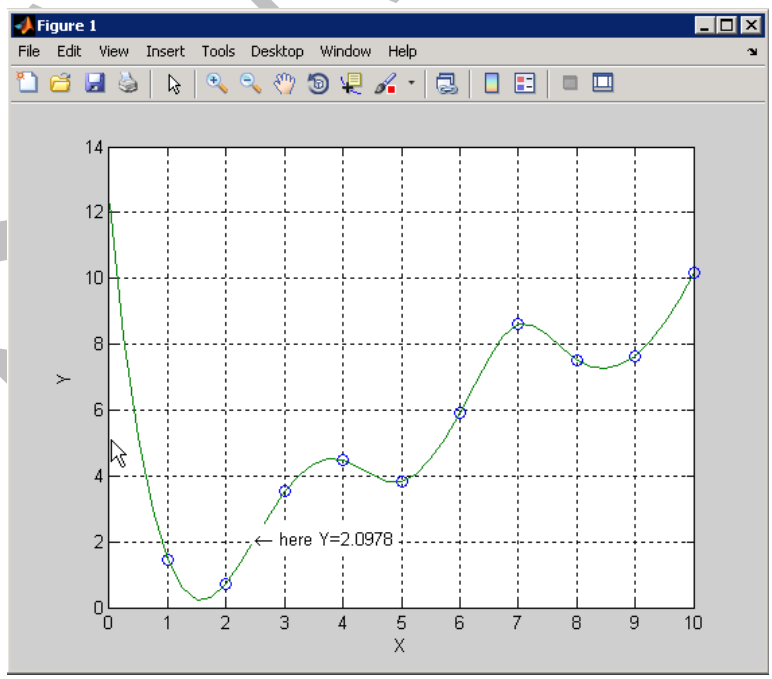
Here is an example with a random vector and the function **spline( )**:

```
x = 1:10;
y = x + randn(1,10);
xx = 0:.25:10;
spline_int = spline(x,y,xx);
plot(x,y,'o',xx,spline_int );

%get a value at a given point
point_of_interest=2.5;
Y_spline=spline(x,y,point_of_interest);

text(point_of_interest,Y_spline,...
    strcat('\leftarrow here Y=',num2str(Y_spline)),...
    'backgroundColor','w');
grid on;
xlabel('X'),ylabel('Y');
```

The result will be:

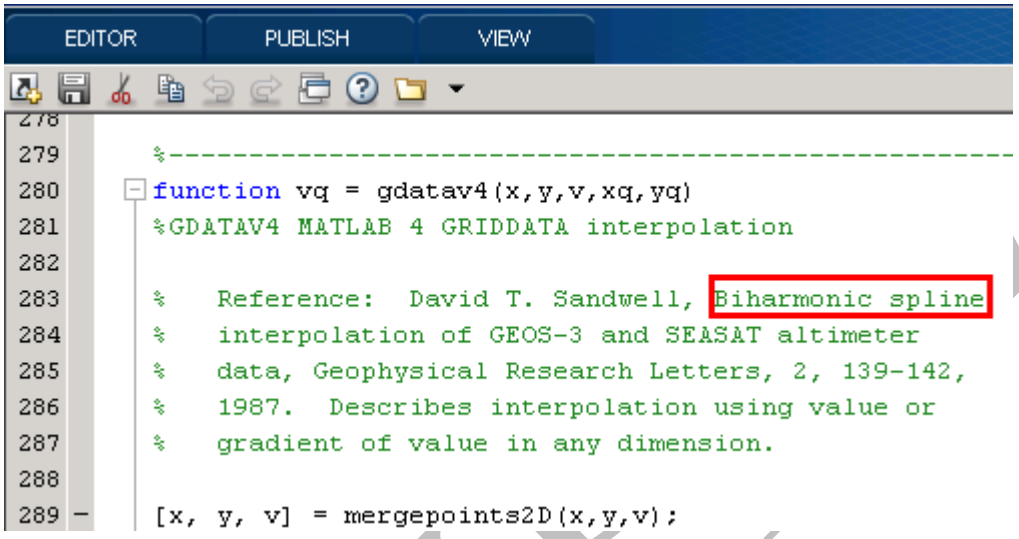


Note: For surface spline interpolation see page 424.

Spline multivariate interpolation

At the page we already saw different technique to interpolate non-uniform vectors measurement points (cubic and Delaunay)

Now have a look to the comparison of the cubic method and spline method call v4 in MATLAB because as described in the source code of the **griddata( )** function we have:



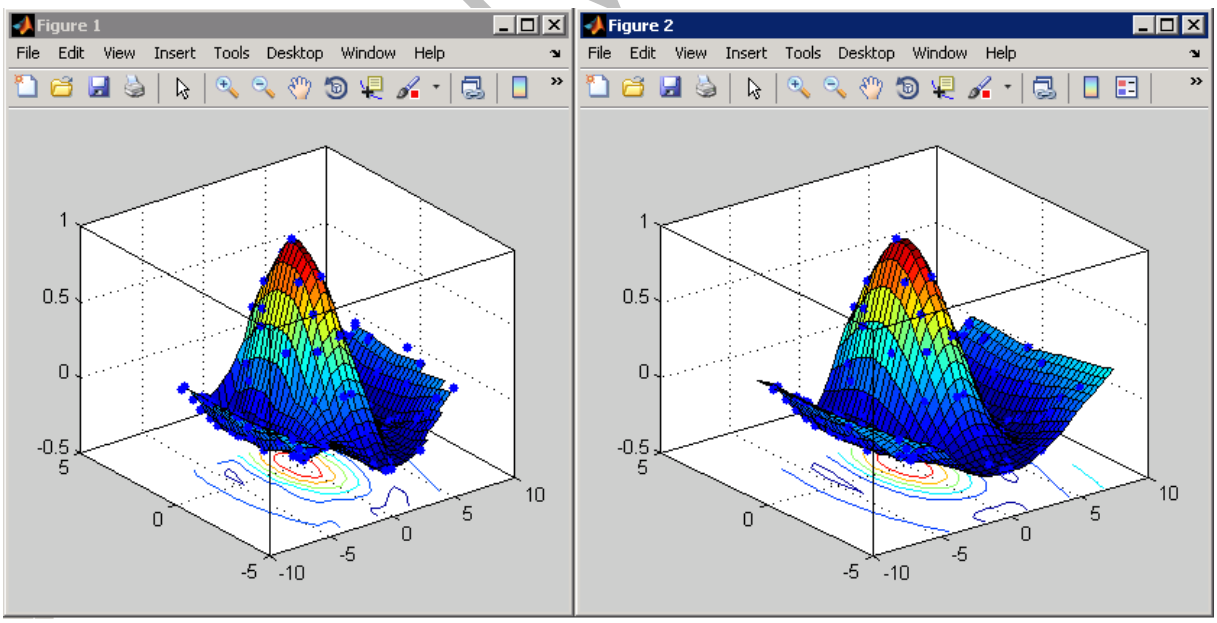
Let us see a comparison example:

SAMPLE

```
EDITOR  PUBLISH  VIEW

1  -    close all;clear all;clc
2
3  -    x=rand(100,1)*16-8;
4  -    y=rand(100,1)*7-5;
5  -    z=sin(sqrt(x.^2+y.^2))./(sqrt(x.^2+y.^2)+eps);
6
7  -    figure(1) %for cubic method
8  -    xlin=linspace(min(x),max(x),30);
9  -    ylin=linspace(min(y),max(y),30);
10
11 -    [X,Y]=meshgrid(xlin,ylin);
12 -    Z=griddata(x,y,z,X,Y,'cubic');
13 -    surfc(X,Y,Z);
14 -    hold on;
15 -    plot3(x,y,z,'.','MarkerSize',15);
16 -    box on;
17
18 -    figure(2) %for spline method
19 -    [X,Y]=meshgrid(xlin,ylin);
20 -    Z=griddata(x,y,z,X,Y,'v4');
21 -    surfc(X,Y,Z);
22 -    hold on;
23 -    plot3(x,y,z,'.','MarkerSize',15);
24 -    box on;
```

This will give (left cubic, right spline):



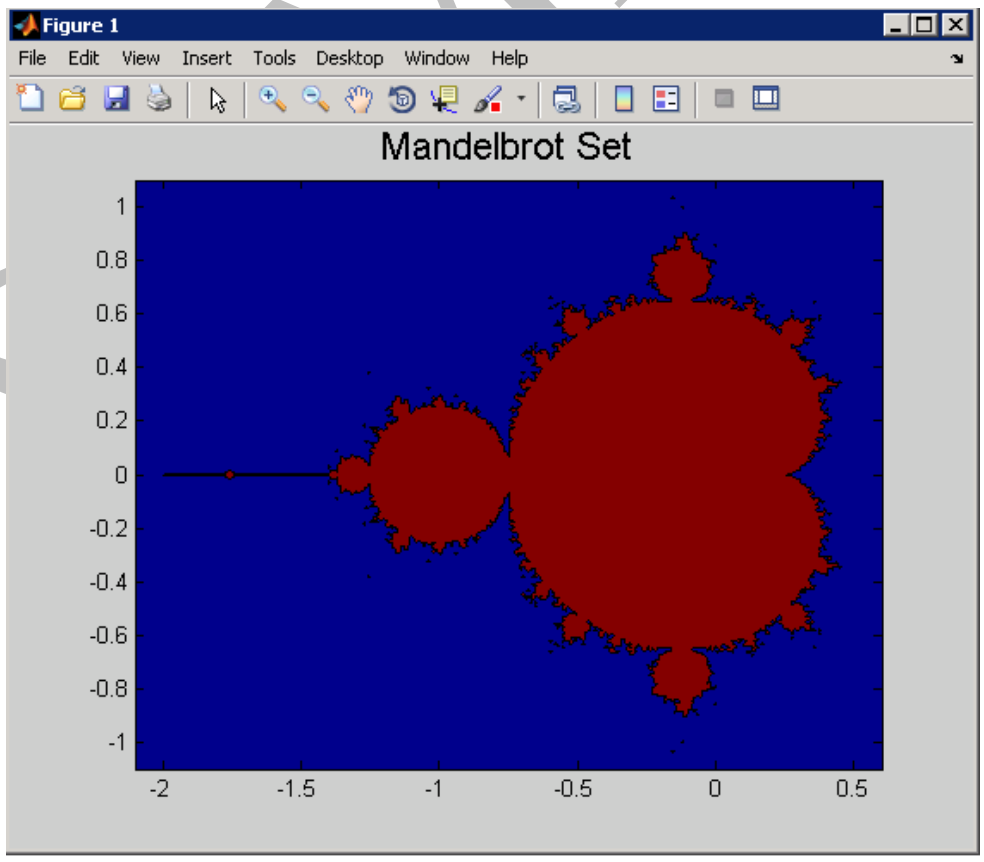
As you can see the spline method is more smoothed.

Fractals

Fractals are used by some financial institutions to model volatility and some CGI algorithms (for Special FX movies production). Then it seems to me useful to first go back to the basics of fractals with the Mandelbrot set:

```
h=waitbar(0,'Computing...');
x=linspace(-2.1,0.6,301);
y=linspace(-1.1,1.1,301);
[X,Y]=meshgrid(x,y);
C=complex(X,Y);
Z_max=1e6;it_max=50;
Z = C;
for k = 1:it_max
    Z = Z.^2 + C;
    waitbar(k/it_max)
end
close(h)
contourf(x,y,abs(Z)<Z_max,1)
title('Mandelbrot Set','FontSize',16)
```

The result will be:



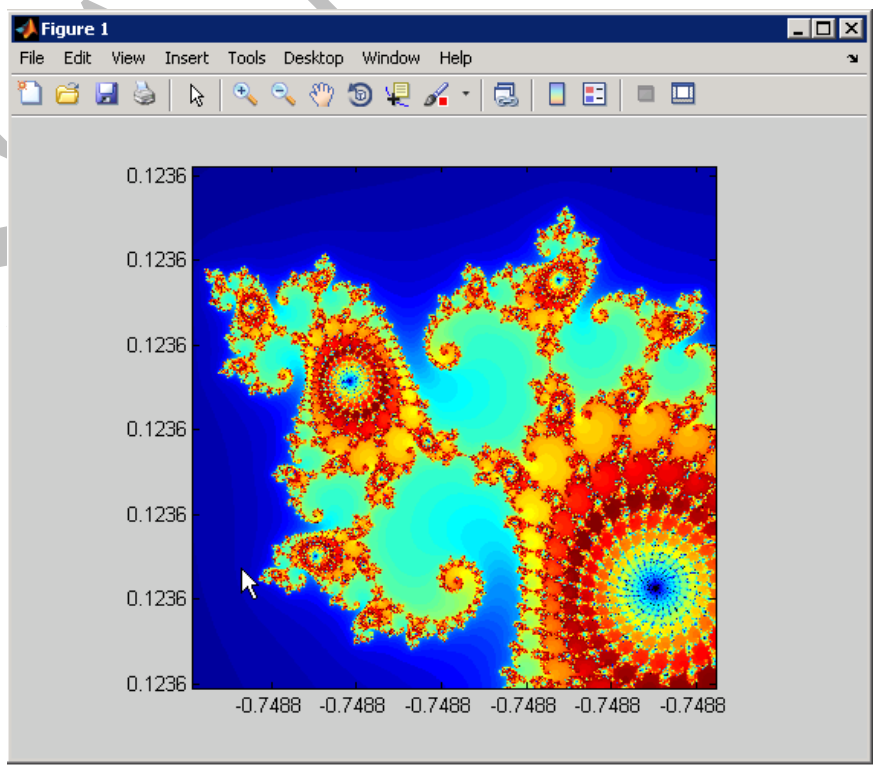
This last figure is a little bit too much old style. Here is a much more pretty way to have the results:

```
maxIterations=500;
gridSize=1000;
%we can play with those limits
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862,  0.123640851045266];

x=linspace( xlim(1), xlim(2), gridSize );
y=linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid(x,y);
z0=xGrid+1i*yGrid;
count=zeros(size(z0));

z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside=abs(z)<=10;
    count=count+inside;
end
count = log( count+1 );

imagesc(x,y,count);
axis image
colormap( [jet();flipud( jet() );0 0 0] );
```



Save chart as image

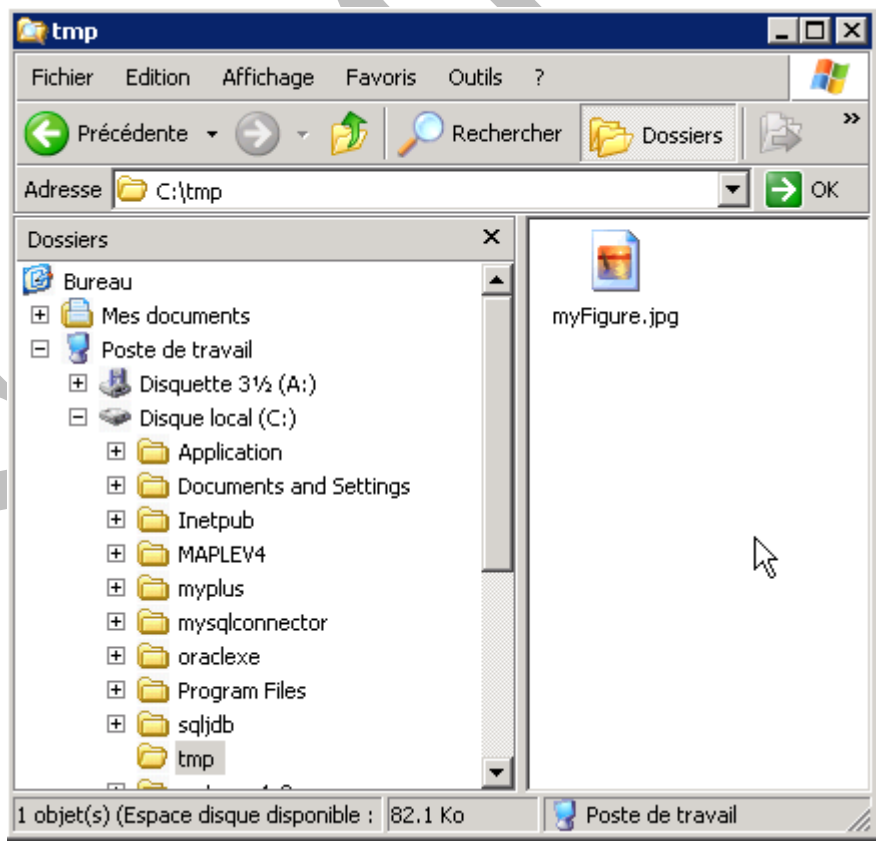
Save on drive

Instead of saving a chart manually as an image it is much more interesting to automate the process. For this purpose we use the command **saveas( )**:

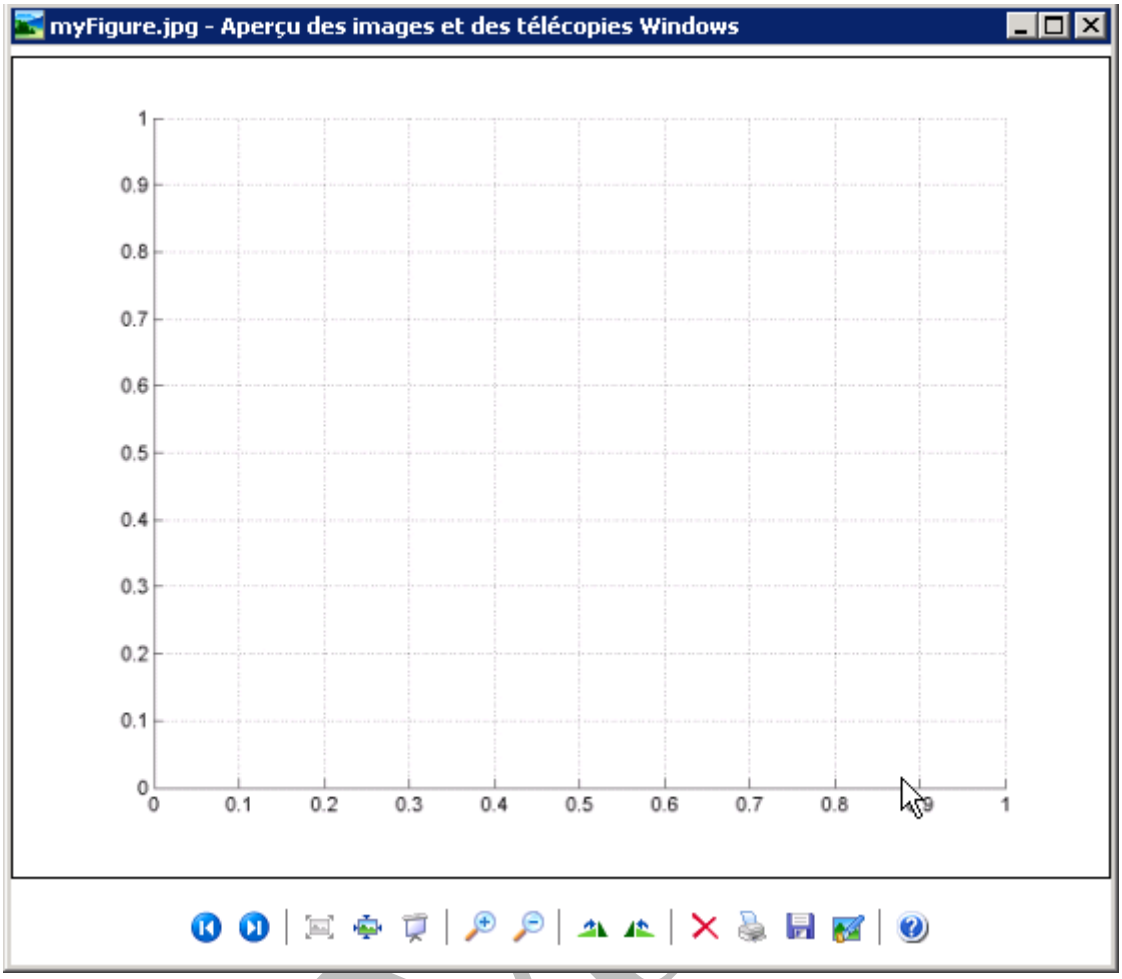
```
EDITOR  PUBLISH  VIEW

1 -  close all;clear all;clc
2 -  f=figure
3 -  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4        'Color',[0.8314 0.8157 0.7483]);
5
6 -  grid on;
7    saveas(f,'C:\tmp\myFigure.jpg');
8
9 -  set(gca,'Xcolor',[0.85 0.85 0.85]);
10 - set(gca,'Ycolor',[0.85 0.85 0.85]);
11 - setAxes=copyobj(gca,gcf);
12 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
13 - MyBox=uicontrol('style','text');
```

This will create a file:



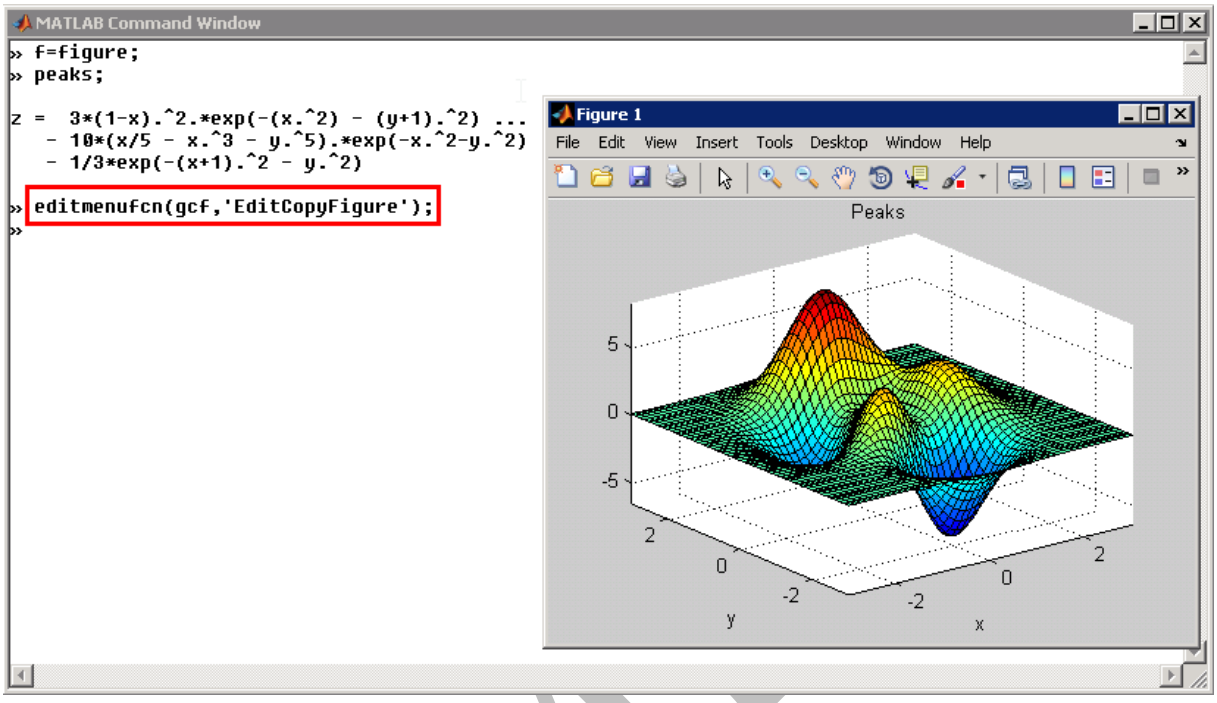
With the following content (empty chart because nothing was plotted):


myFigure.jpg - Aperçu des images et des télécopies Windows
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1

SAMPLE

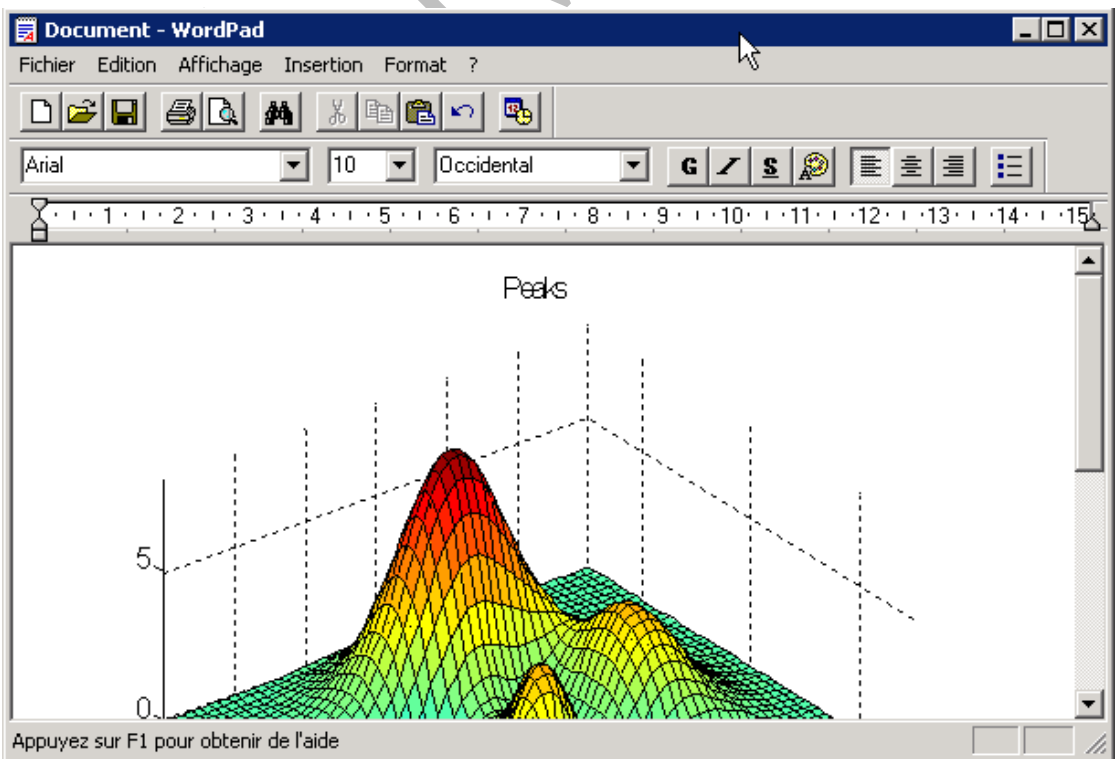
Copy in the clipboard

Most of time the menus and buttons of a chart figure are removed from the figure Windows. Then we have to found also a way to automatize the action to take the figure in the clipoard. For this purpose consider the small following example:



The whole trick resides in the use of the callback function **editmenufcn( )**.

After the user just has to open the software he wants and make a Ctrl+V:

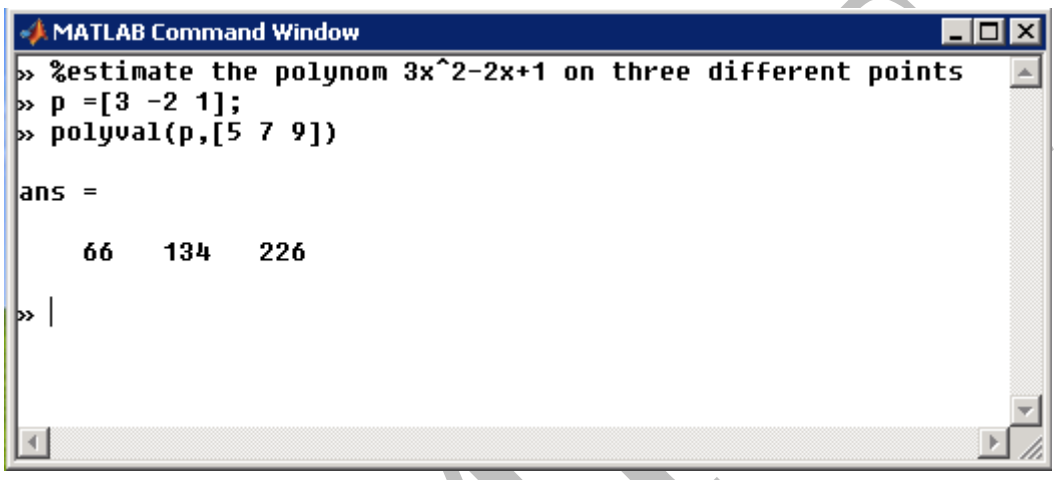


Analysis

This chapter is dedicated to topics related to "math analysis" field. It concern almost all areas of physics, mathematics, engineering, finance, signal (audio/video) processing, etc.

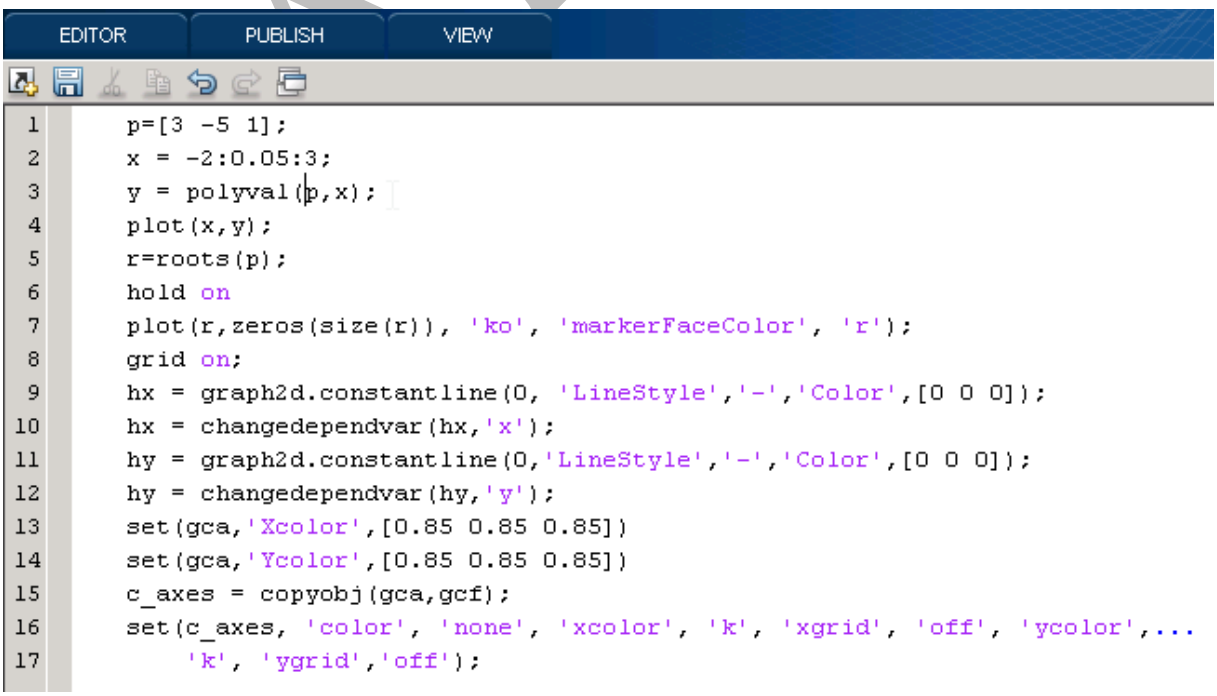
Estimate value of an univariate polynomial on various points

Let us begin with something very easy (high school level) that we already have discovered during our study of MATLAB chart. The estimation of a value of a polynomial on one or many points using the function **polyval( )**:



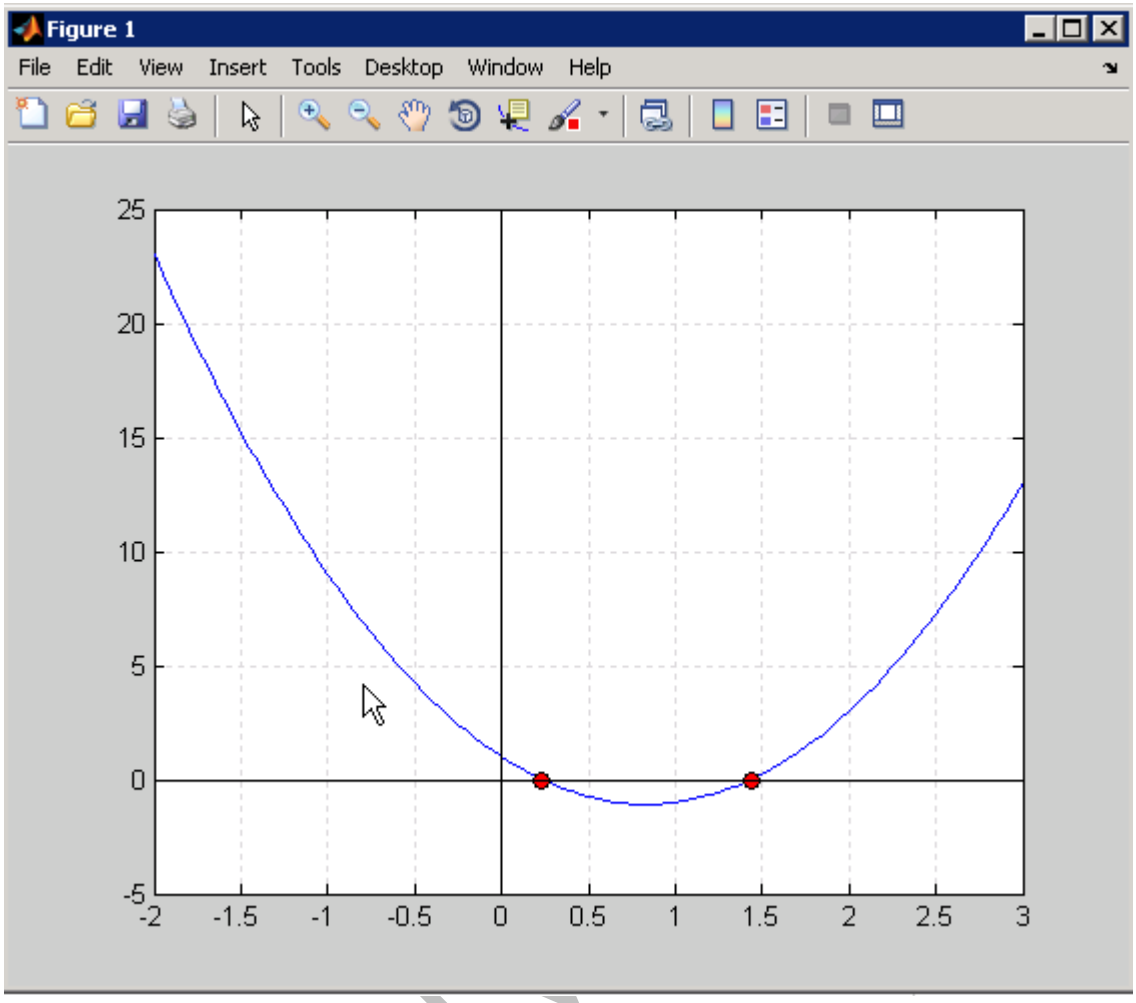
Calculate the roots of an univariate polynomial

Always at high school level and also something we already used during our study of MATLAB chart, how to calculate the roots of a polynomial. There a two methods to do this. The first one is visual and use the functions **polyval( )** and **roots( )**:



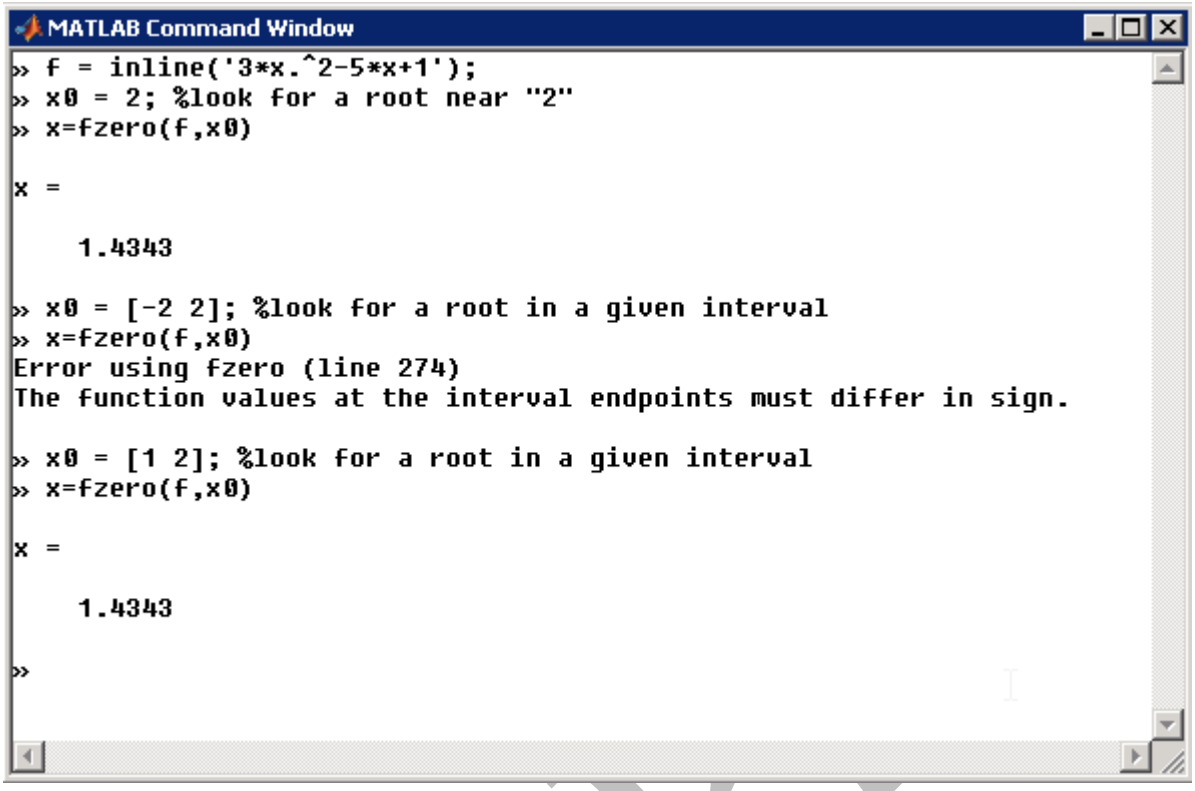

```
1   p=[3 -5 1];
2   x = -2:0.05:3;
3   y = polyval(p,x);
4   plot(x,y);
5   r=roots(p);
6   hold on
7   plot(r,zeros(size(r)), 'ko', 'markerFaceColor', 'r');
8   grid on;
9   hx = graph2d.constantline(0, 'LineStyle','-','Color',[0 0 0]);
10  hx = changedependvar(hx,'x');
11  hy = graph2d.constantline(0,'LineStyle','-','Color',[0 0 0]);
12  hy = changedependvar(hy,'y');
13  set(gca,'Xcolor',[0.85 0.85 0.85])
14  set(gca,'Ycolor',[0.85 0.85 0.85])
15  c_axes = copyobj(gca,gcf);
16  set(c_axes, 'color', 'none', 'xcolor', 'k', 'xgrid', 'off', 'ycolor',...
17      'k', 'ygrid','off');
```

This gives us a pretty result to see:

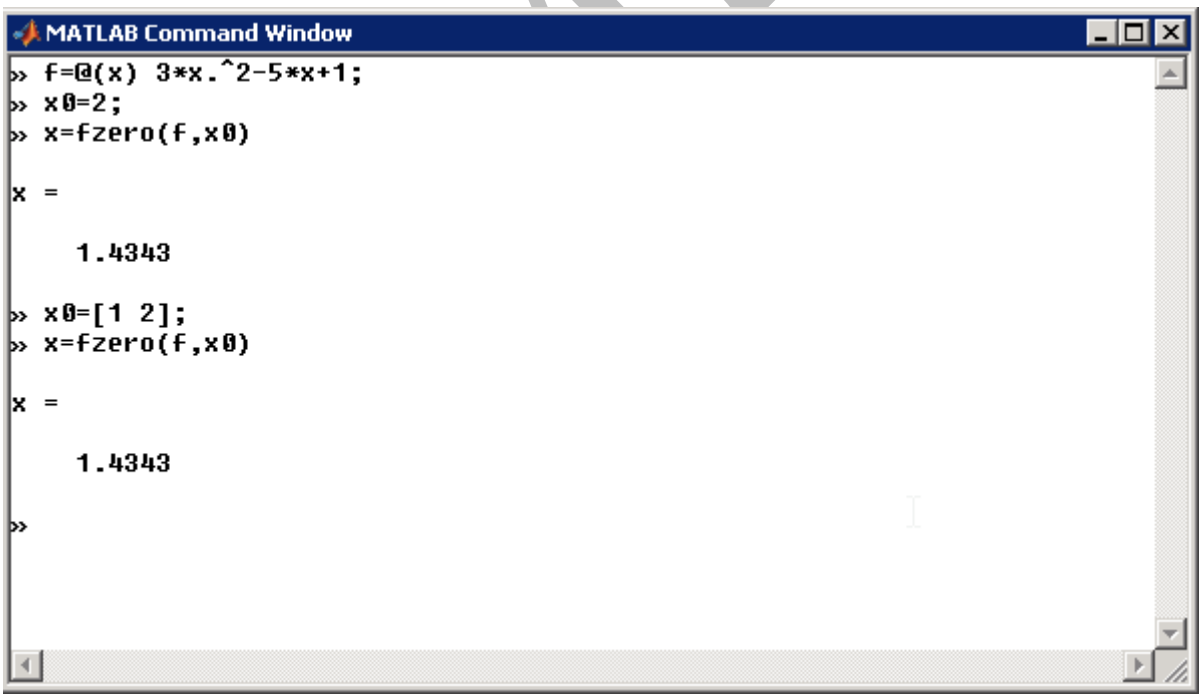


The second method is less visual and more "approximate".

SAM



Second method based on "modern notation" with the function **fzero( )** that don't use Newton-Raphson method FYI...:

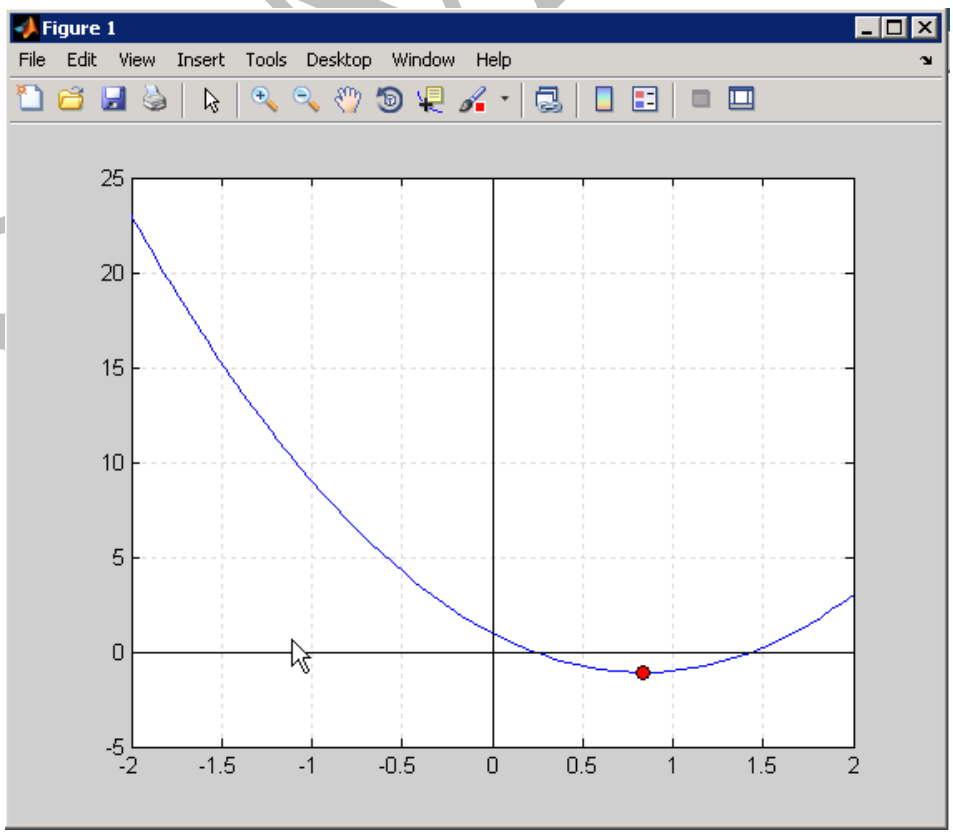


Find the local min/max of an univariate function in a range

The idea here is the same as **fzero( )** at the difference that we search a local minima. Why "local"? Because the function **fminbnd( )**, for "function minimum bounded", search for only one minima and the smallest unique minima and does not return all minimas of a function (there exist free toolbox for this last purpose):

```
f=@(x) 3*x.^2-5*x+1;
x=fminbnd(f, -2, 2);
fplot(f,[-2 2])
hold on
plot(x,f(x),'ko','markerFaceColor','r');
grid on;
hx=graph2d.constantline(0, 'LineStyle','-','Color',[0 0 0]);
hx=changedependvar(hx,'x');
hy=graph2d.constantline(0,'LineStyle','-','Color',[0 0 0]);
hy=changedependvar(hy,'y');
set(gca,'Xcolor',[0.85 0.85 0.85])
set(gca,'Ycolor',[0.85 0.85 0.85])
c_axes = copyobj(gca,gcf);
set(c_axes,'color','none','xcolor','k','xgrid','off','ycolor','k',...
    'ygrid','off');
```

This will give:



In the *Command Window* this will look like:

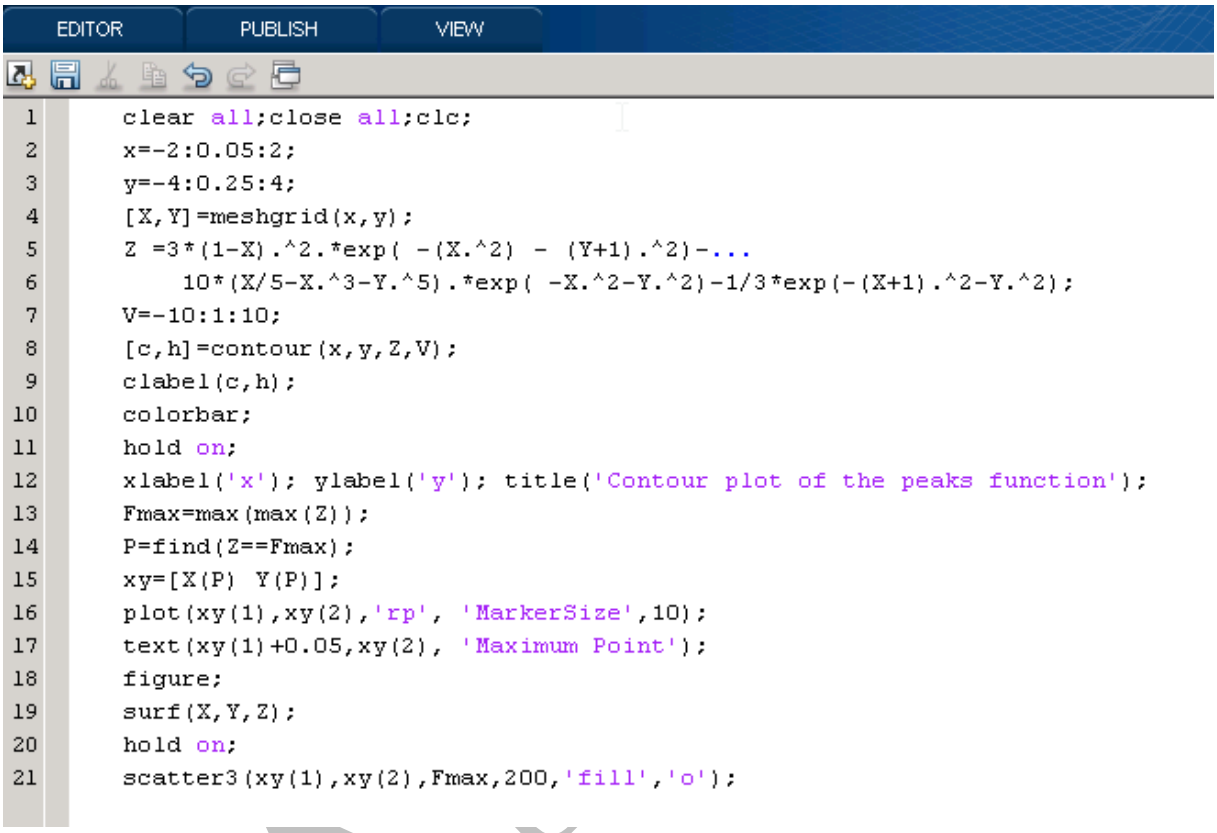


SAMPLE

Find the local min/max of a bivariate function

1st Method (very old one)

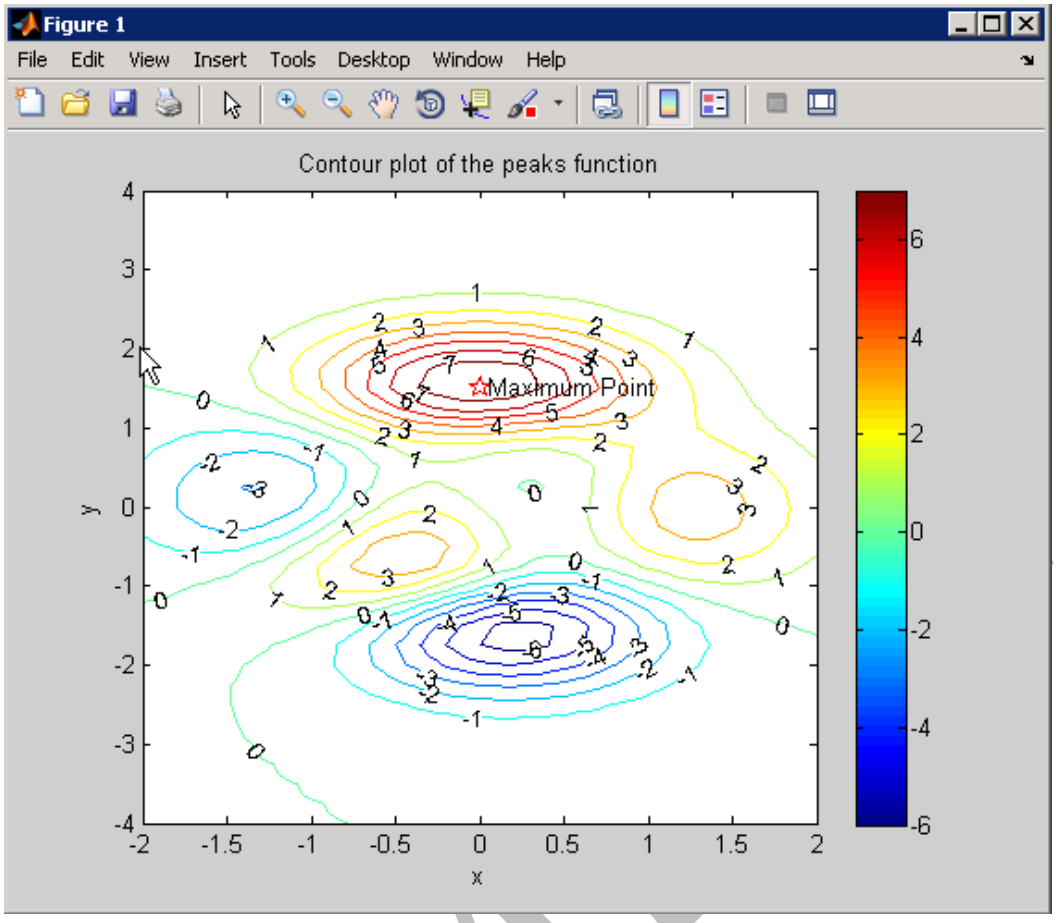
There were no functions at the beginning of MATLAB as far I as I know to get the global minimum/maximum of a bivariate function. Thus we were obliged to use a trick using the **max( )** and **find( )** functions:

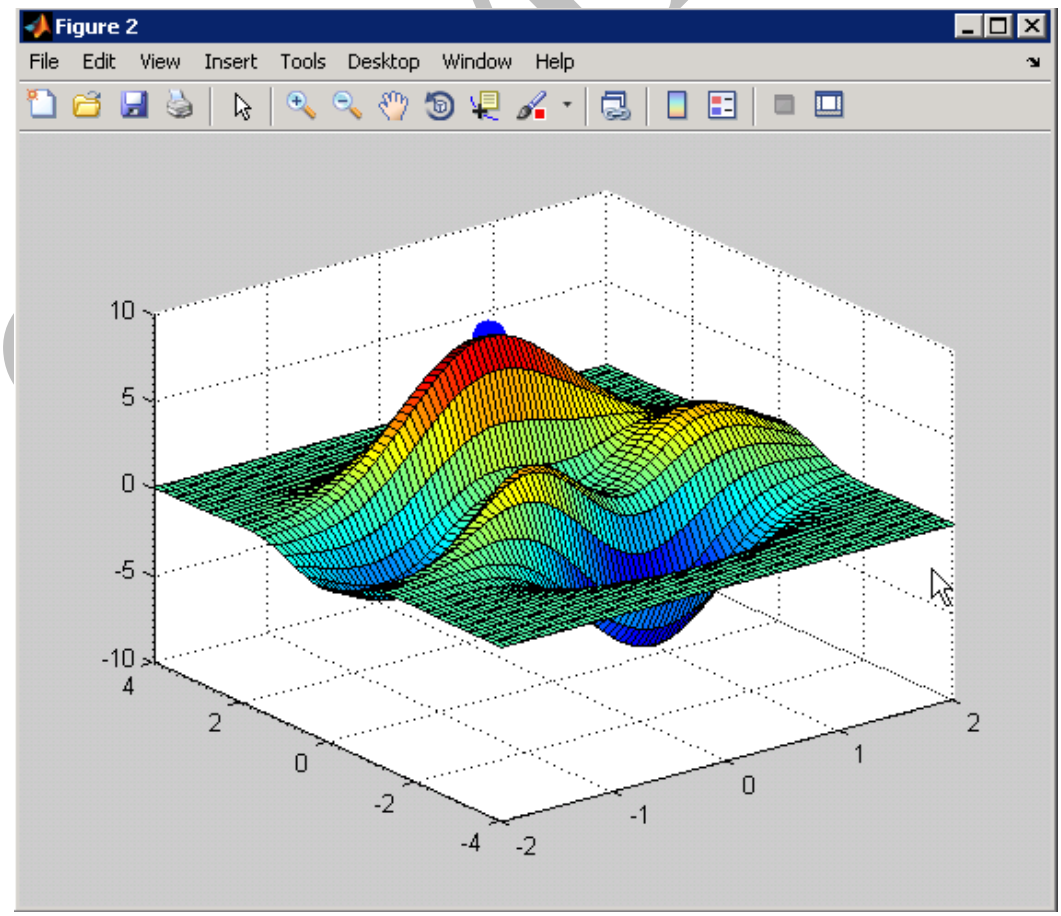



```
clear all;close all;clc;
x=-2:0.05:2;
y=-4:0.25:4;
[X,Y]=meshgrid(x,y);
Z =3*(1-X).^2.*exp( -(X.^2) - (Y+1).^2)-...
    10*(X/5-X.^3-Y.^5).*exp( -X.^2-Y.^2)-1/3*exp(-(X+1).^2-Y.^2);
V=-10:1:10;
[c,h]=contour(x,y,Z,V);
clabel(c,h);
colorbar;
hold on;
xlabel('x'); ylabel('y'); title('Contour plot of the peaks function');
Fmax=max(max(Z));
P=find(Z==Fmax);
xy=[X(P) Y(P)];
plot(xy(1),xy(2),'rp', 'MarkerSize',10);
text(xy(1)+0.05,xy(2), 'Maximum Point');
figure;
surf(X,Y,Z);
hold on;
scatter3(xy(1),xy(2),Fmax,200,'fill','o');
```

The result is visible on the next page with two various possibilities:

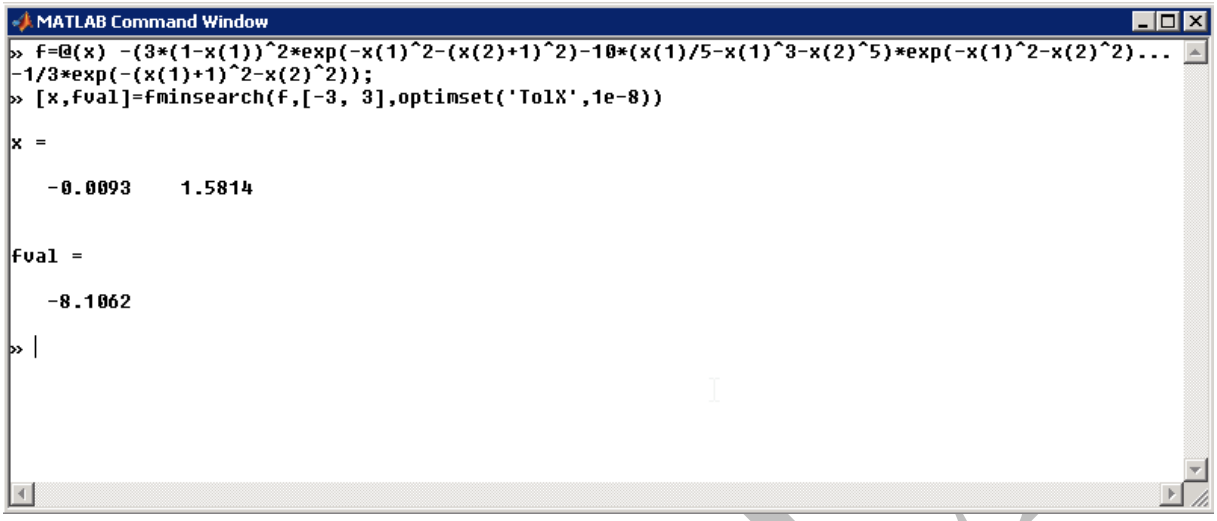
SA


Figure 1
File Edit View Insert Tools Desktop Window Help
Contour plot of the peaks function
Maximum Point
x
y


Figure 2
File Edit View Insert Tools Desktop Window Help

2nd Method (modern one)

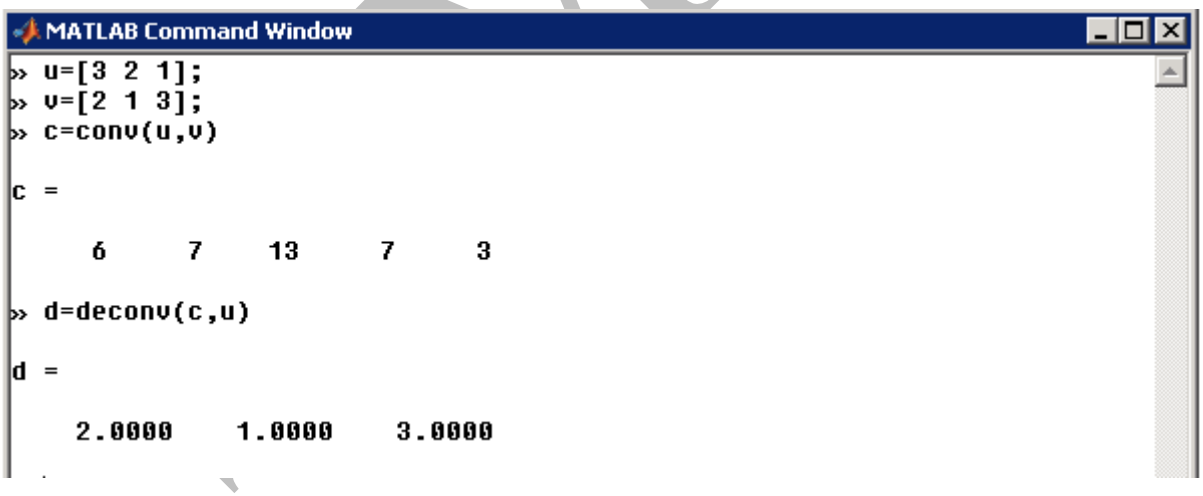
Here we just use the function **fminsearch( )** as following:



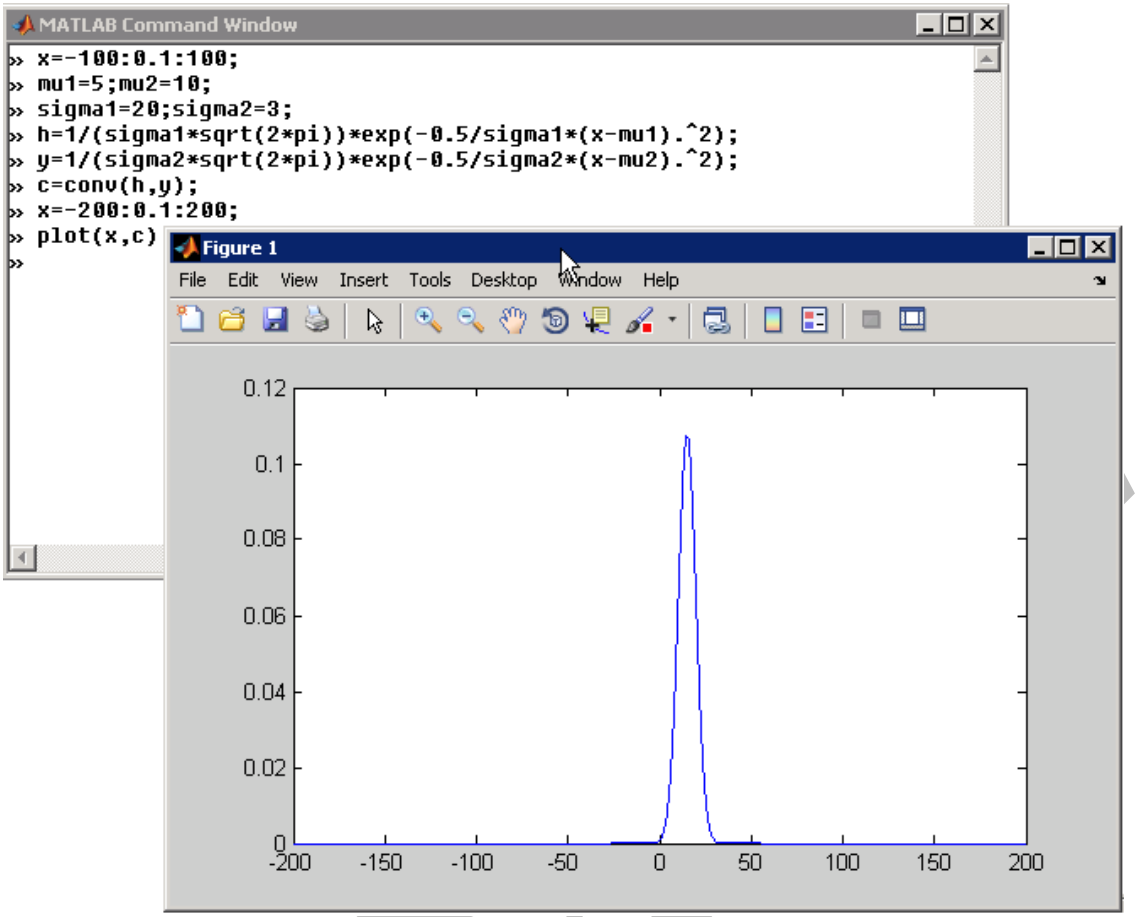
Discrete convolution

Convolution is a used a lot in signal (and also sound/images) processing to modify some original images to get new ones or to add echo/filters to sounds. Here we will two very elementary examples.

First we control the basics with the function **conv( )** and **deconv( )**:

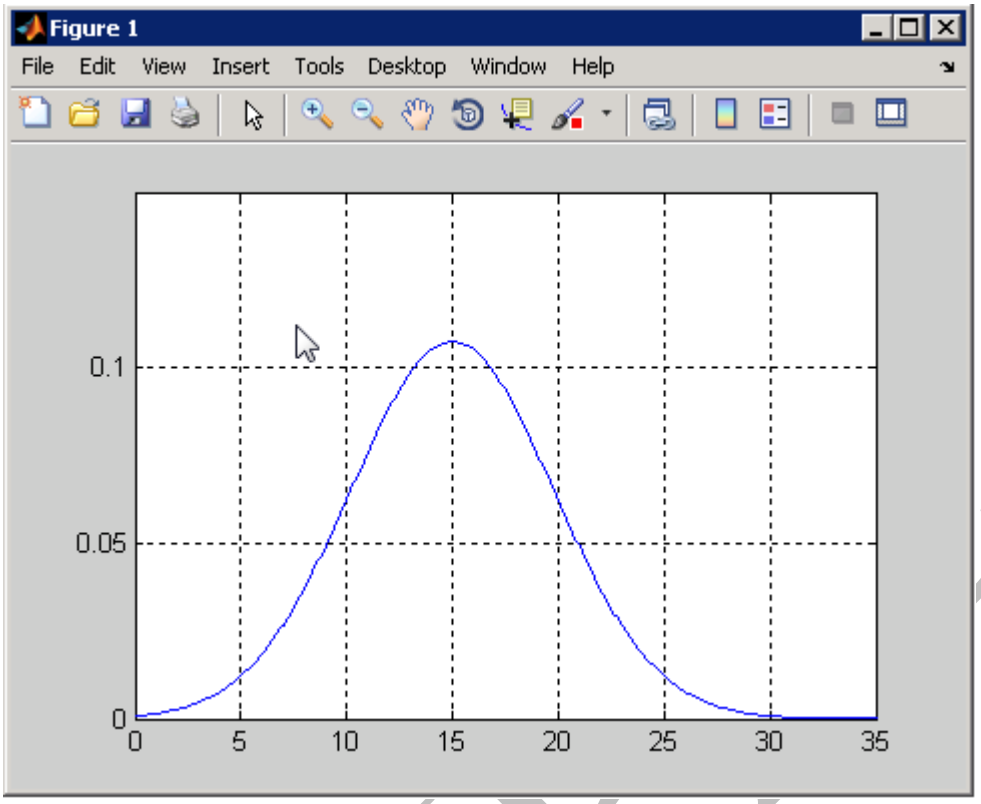


We can also check that the convolution of two Normal random variables is equal to their sum:



And if we zoom , we get:

SAMPLE



We must forget de deconvolution in this case!

A study of the U.S. Army Statistical Department has shown that the convolution of random variable is faster than Monte Carlo but only for two or three random variable. Over 3 random variables, MC is faster.

ToDo: Calculate the mean and variance of this convolution

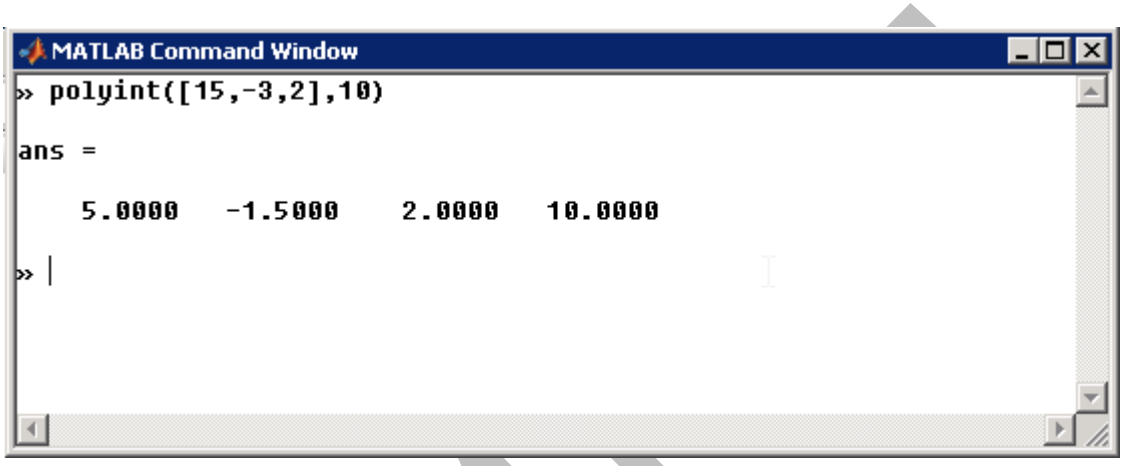
SAMPLE

Integration

Integration is not only useful in physics or in pure mathematics but also a lot in engineering finance. Let us see first some very easy applications and further in the chapter about the Statistical Toolbox we will deepen this field using Monte Carlo technics.

Pseudo-symbolic integration of univariate polynomial

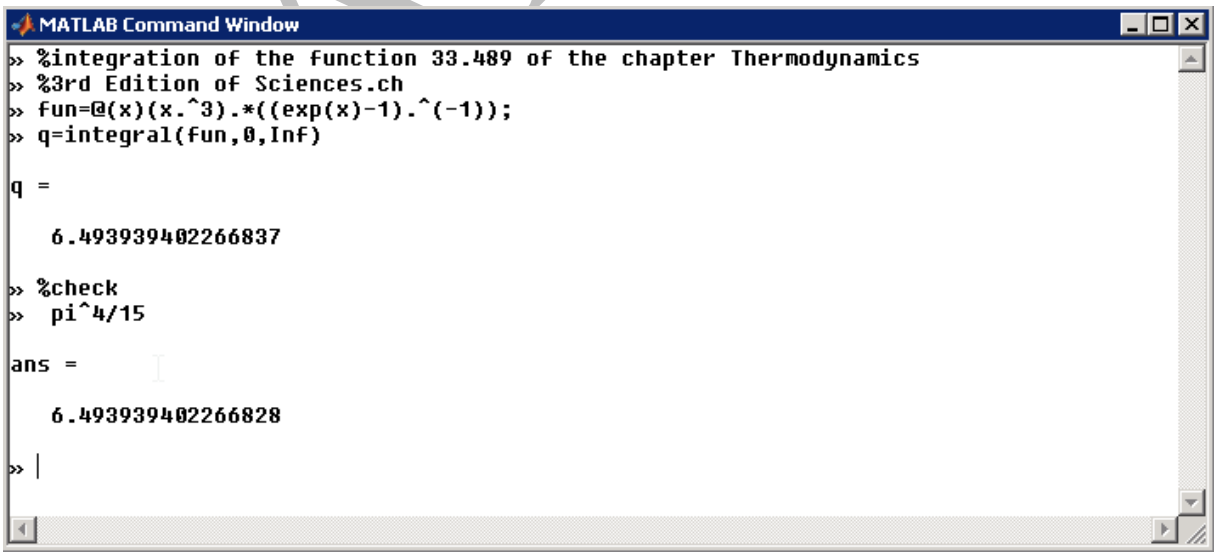
The most easiest integration that can be done in MATLAB is certainly using the function **polyint( )** to integrate a simple polynomial. The last argument is the constant of integration...:



Sometimes I think that this function is really a joke...

Numeric univariate integration

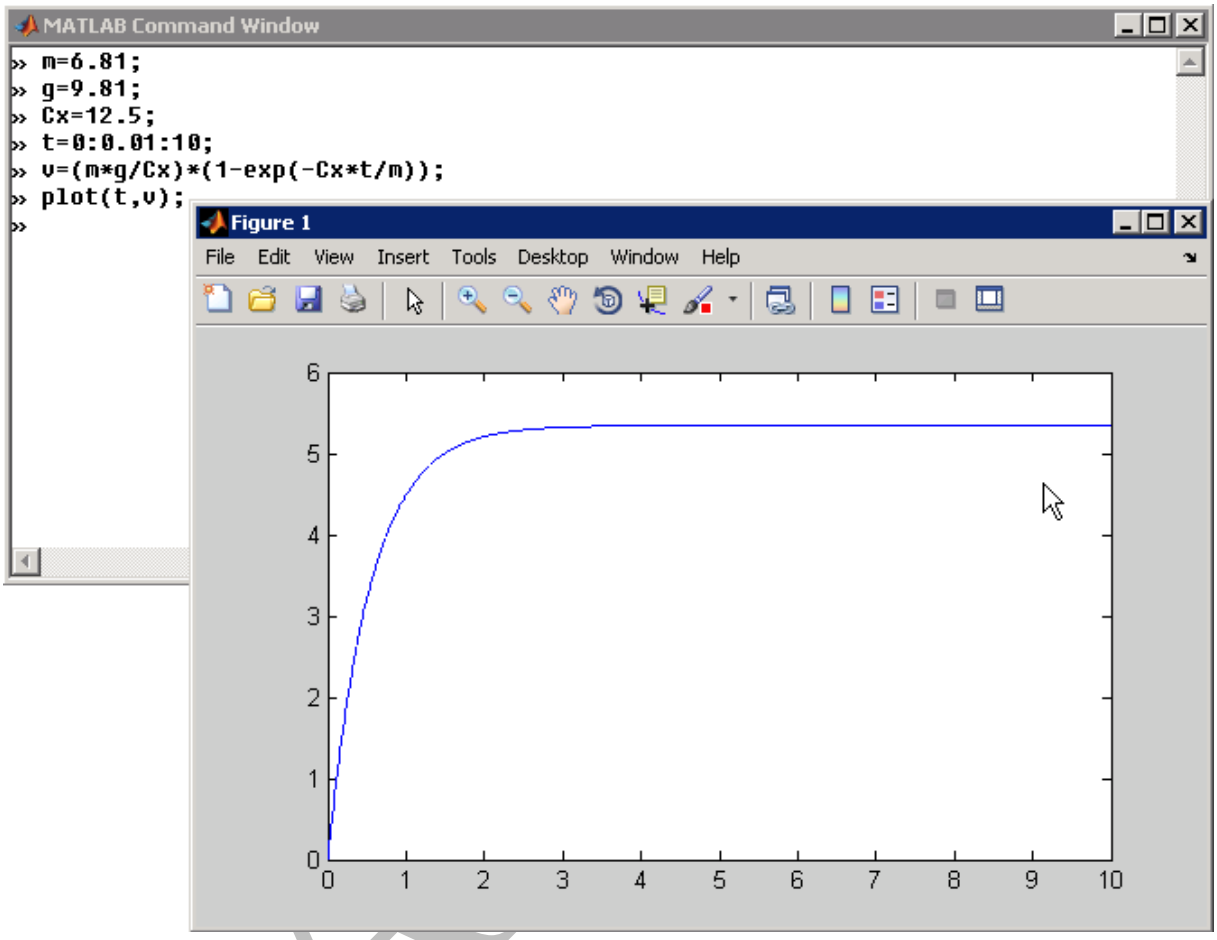
Here it begins to be much more interesting! We the function **integral( )** we can calculate the integral of many definite functions. We will use an example that made us a lot of trouble in the theoretical chapter of Thermodynamics:



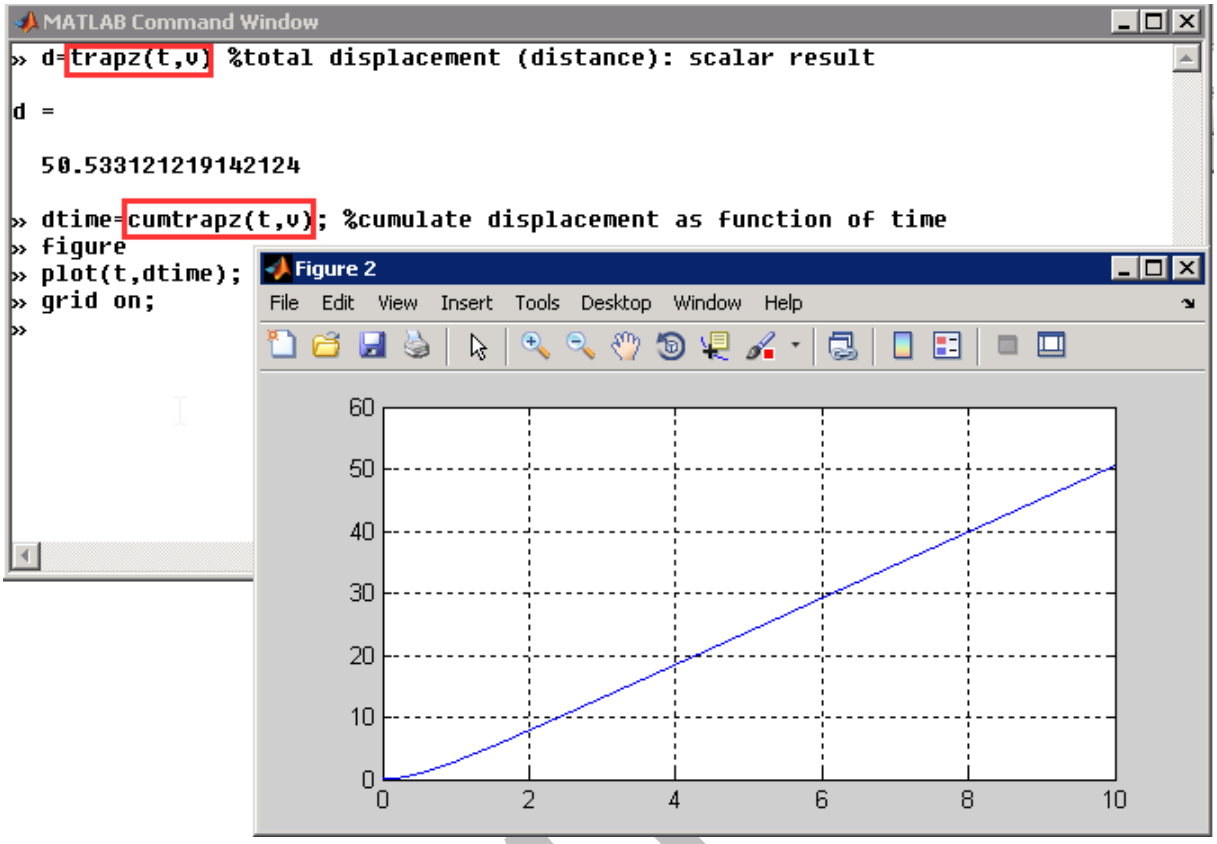
Numeric trapezoidal univariate integration

Even more important: if the function is not explicitly known, we can integrate a vector defined function.

For the example consider the famous academic following example:

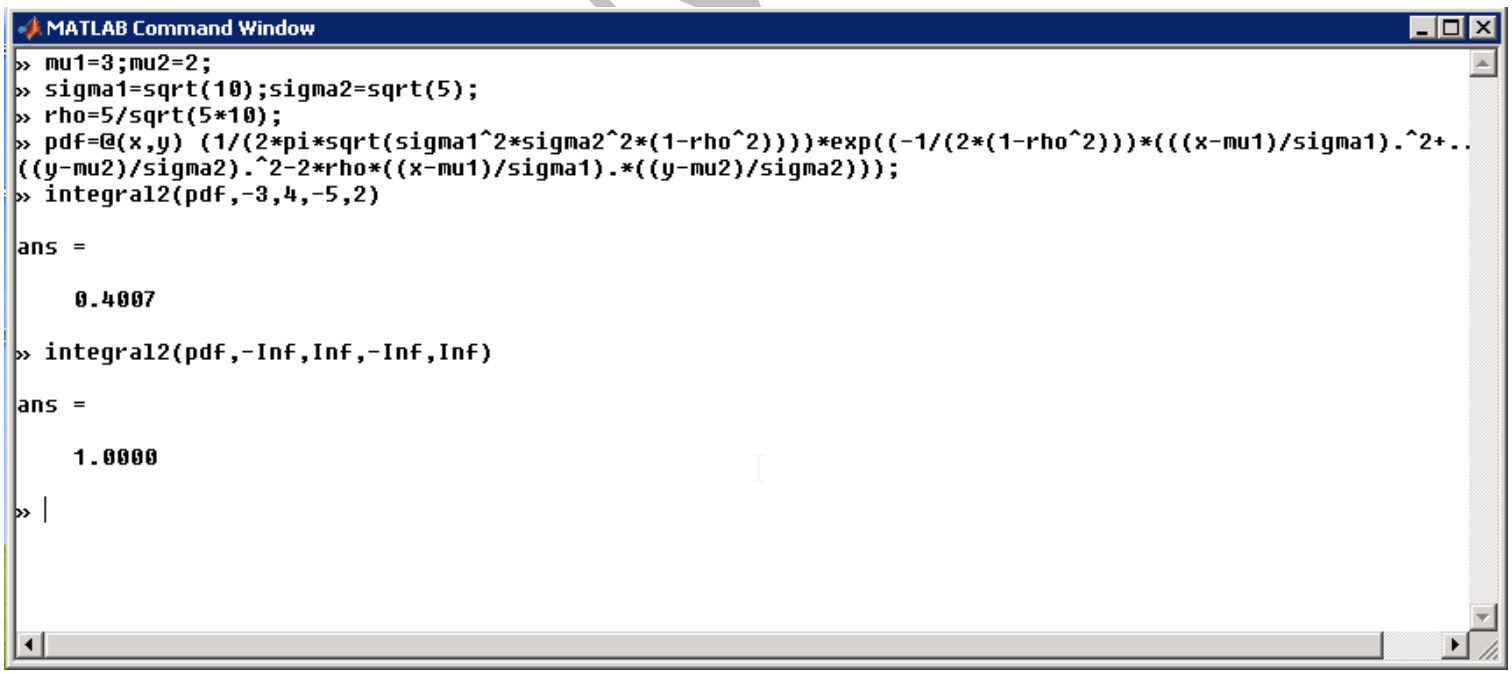


And now we calculate the total or cumulated displacement over the time by integration with **trapz( )** and **cumtrapz( )** thanks to the fact that we know the velocity at each moment:



Double numeric integration

Use a lot in statistics and finance the double integration for explicitly know functions can be done thanks to **integral2( )**:



Or using the trapezoidal method (for those having an old MATLAB version):

```
1  f= @(x,y) (1/(2*pi*sqrt(sigma1^2*sigma2^2*(1-rho^2))))*exp((-1/(2*(1-rho^2)))*(((x-mu1)/sigma1).^2+((y
2  rho=5/sqrt(5*10);
3  mu1=3;mu2=2;
4  sigma1=sqrt(10);
5  sigma2=sqrt(5);
6  N = 2000;
7  xpts1d = linspace(-3,+4,N+1);
8  ypts1d = linspace(-5,+2,N+1);
9  xpts = xpts1d'*ones(1,N+1);
10 ypts = ones(N+1,1)*ypts1d;
11 values = f(xpts,ypts);
12 format long g
13 I = trapz(xpts1d,trapz(ypts1d,values,2),1)
```

So we get:

```
Command Window

I =

   0.400659680067849

fx >>
```

Derivation

Derivation is with MATLAB mainly used in mechanics or physics and only sometimes in finance. We will see now some typical function to calculate derivate of explicit expressions or vector measures.

Pseudo-symbolic derivation of polynomial

The most easiest integration that can be done in MATLAB is certainly using the function **polyder( )** to derivate a simple polynomial:

```
MATLAB Command Window
>> polyder([15,-3,2])

ans =

    30    -3

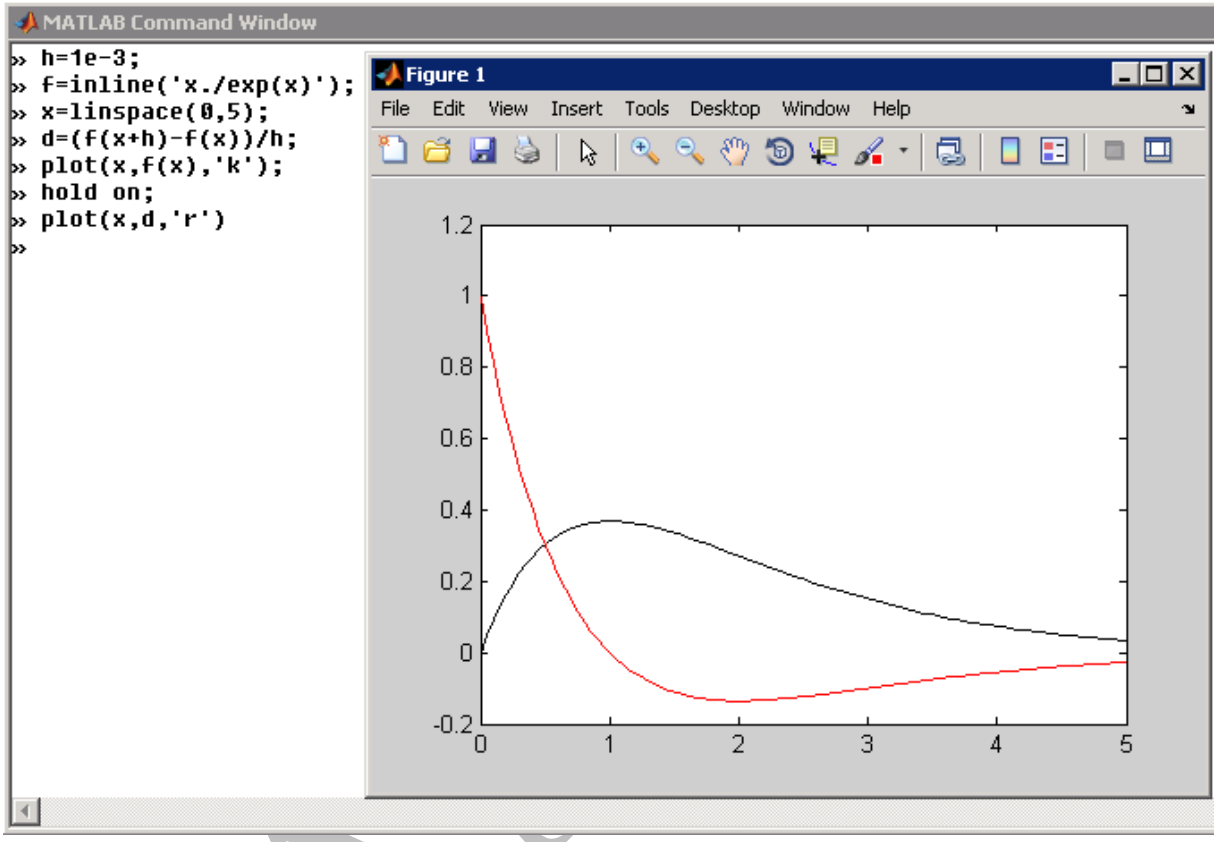
>>
```

Sometimes I think also that this function is really a joke...

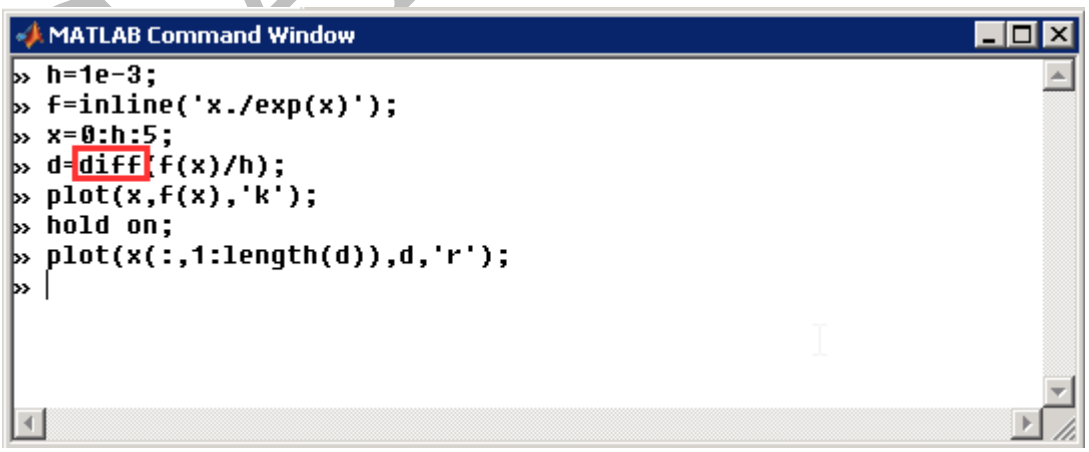
Numeric derivation

There a different way to do a numeric derivation in MATLAB (like in many others softwares) and it is always interesting to study this subject of derivation that has a lot of subtilities.

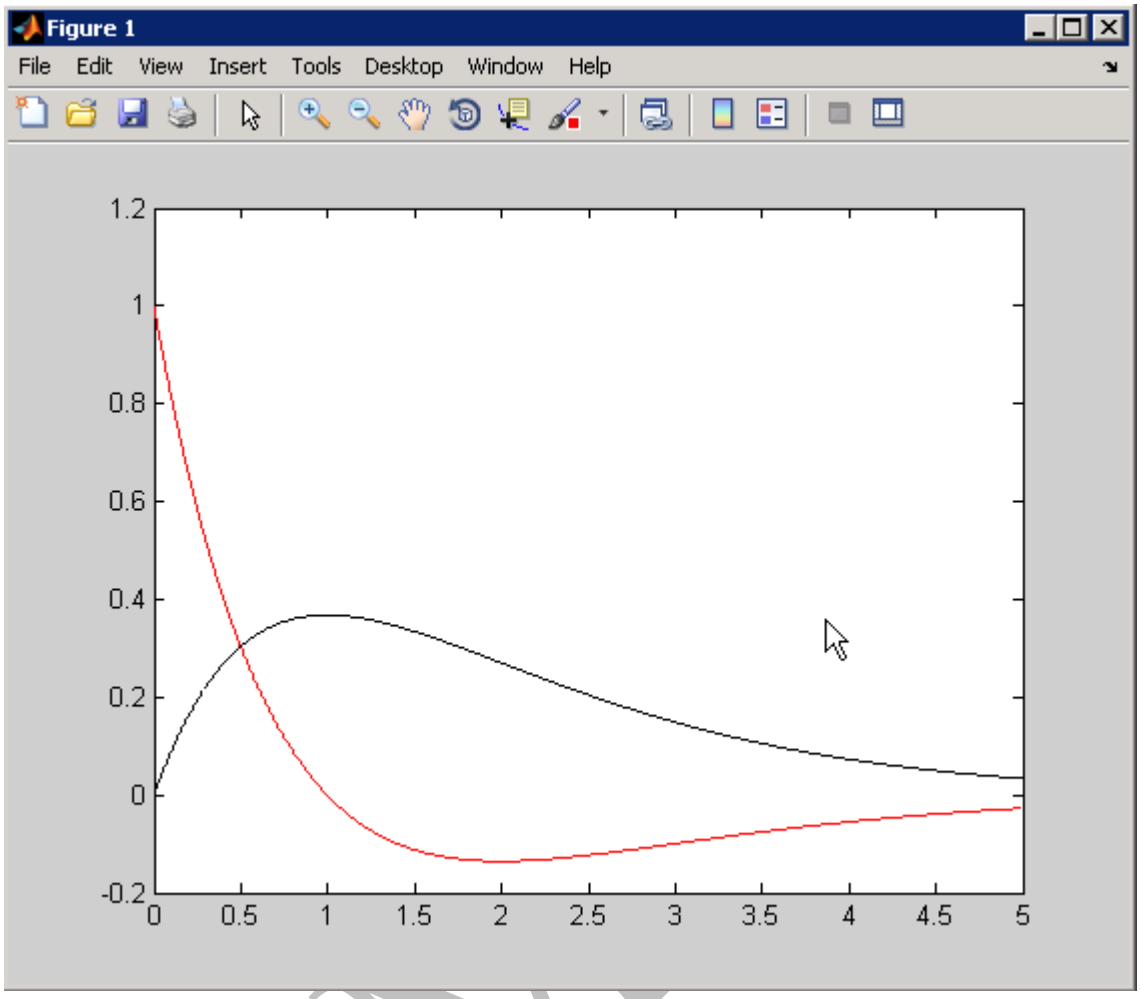
First begin with a simple right derivate with a large accuracy of 1e-3 and using no special functions:



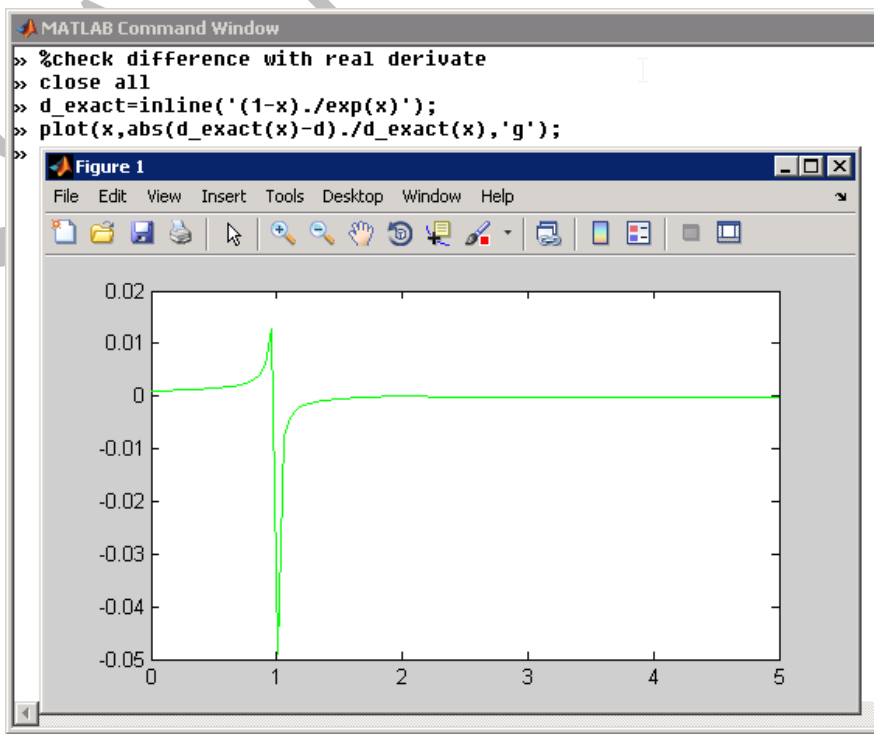
Now see the "official" method for derivate using the function **diff( )**...:



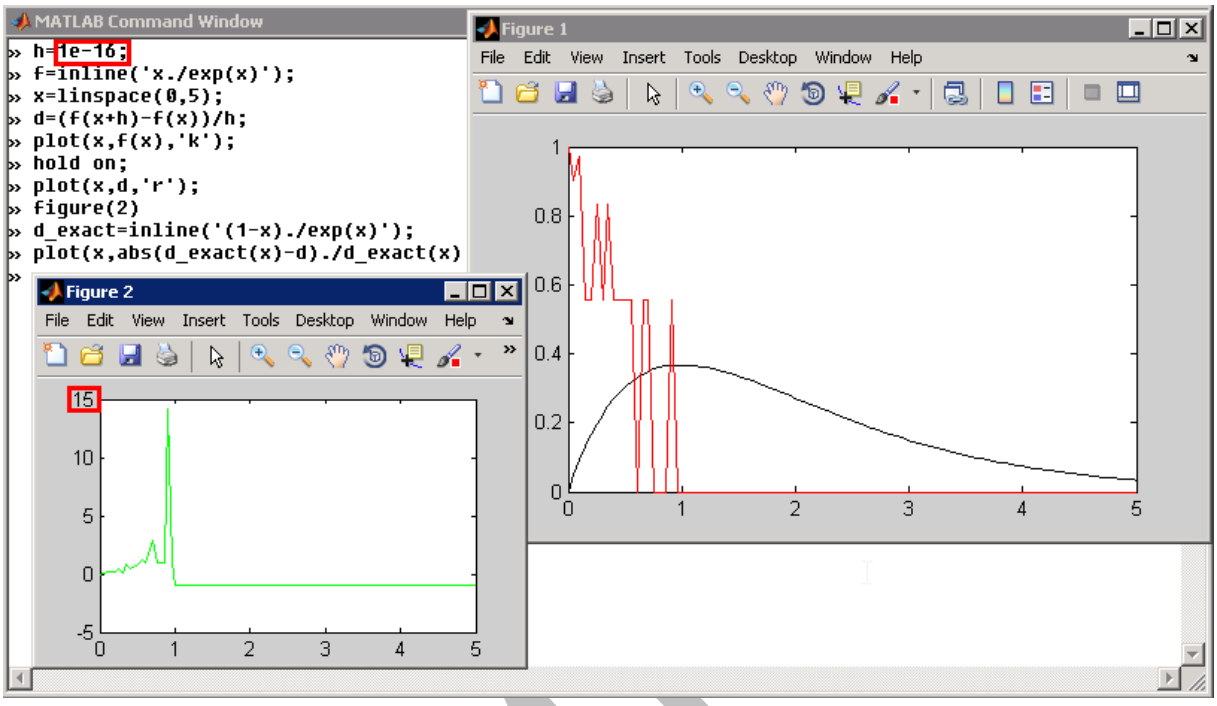
That gives:



Now see the difference between the reald derivate and the one calculate numerically:



Take care! Due to rounding error and truncatanation this method is not accurate for very small values of *h* (of the order of *eps*) for any function! (the smallest is not always the better...) as you can see below:



SAMPLE

Differential equations (ODE/PDE)

Like everything else in the e-book, all the examples here are taken from my e-book "*Opera Magistris – Element of Applied Mathematics for Engineers*" where you can found the detailed proofs of the different ODE/PDE solutions.

Before to see the example you must know that it seems to be an obligation to have the ODE/PDE in an external *.m file.

Pendulum (ODE)

Let us begin with the pendulum ODE. Because the approach in MATLAB is far different for the notation of pure mathematics we will write the equations to make you understand the thinks better (we hope so...).

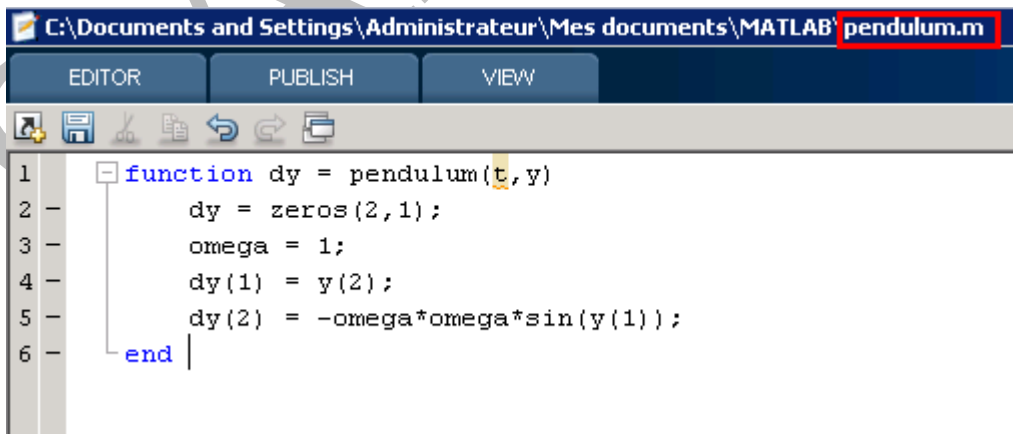
Remember that for the simple pendulum we get in the Mechanic theoretic course for small angles the following 2nd order nonlinear ODE (the sign in front of the sinus doesn't really matter):

$$\frac{d^2\alpha}{dt} = -\omega_0^2 \sin\alpha \underset{\substack{x=\alpha \\ y=\frac{dx}{dt}}}{\Leftrightarrow} \frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ -\omega_0^2 \sin x \end{bmatrix}$$

with the initial condition that *x* and *y* are at the beginning both equal to zero. In MATLAB we need to rewrite this equation into two first order equations.

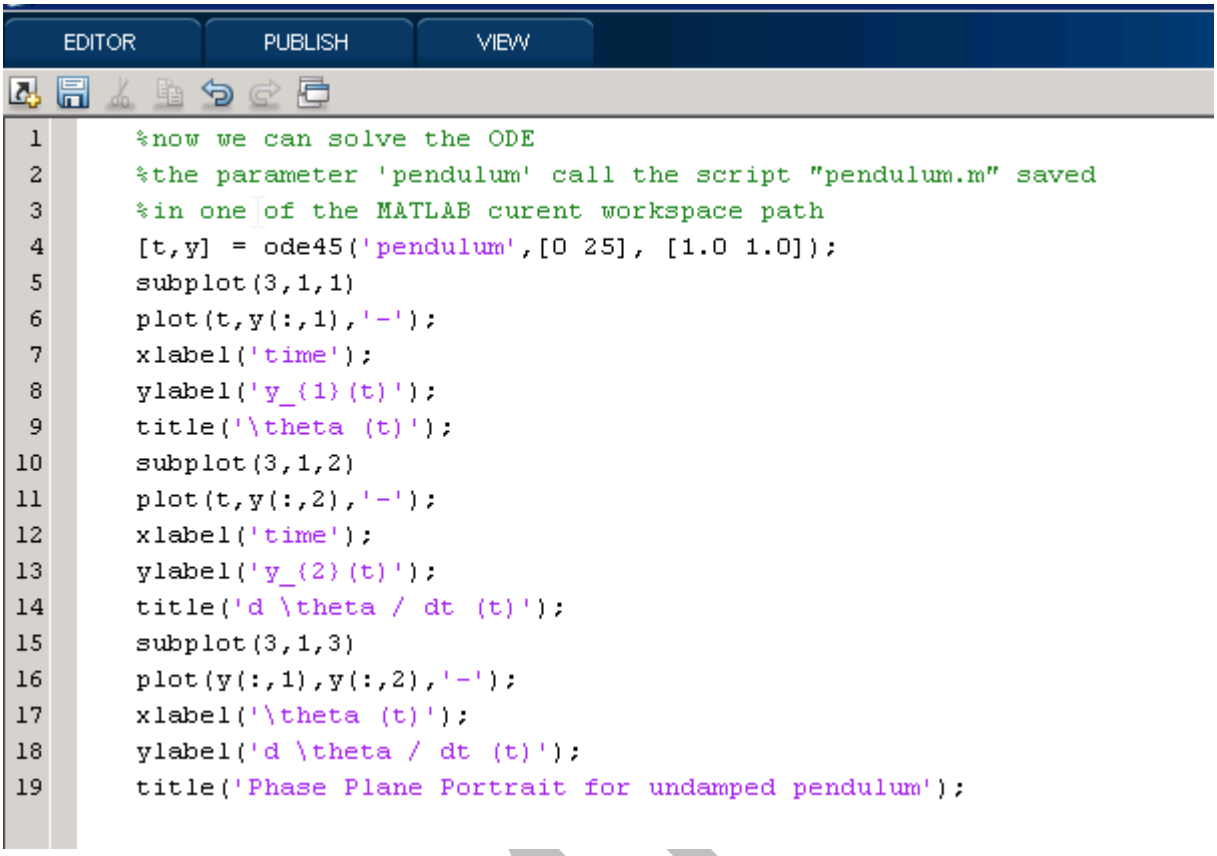
MATLAB ODE function will always, as far as I know, always return the components of the **red** vector.

In MATLAB we have to write the ODE system in a specific file and define the initial values of the derivate:



And after only we can solve it between times [0, 25] and initial position and pulsation using the **ode45( )** solver without any special parameters:

SAMPLE



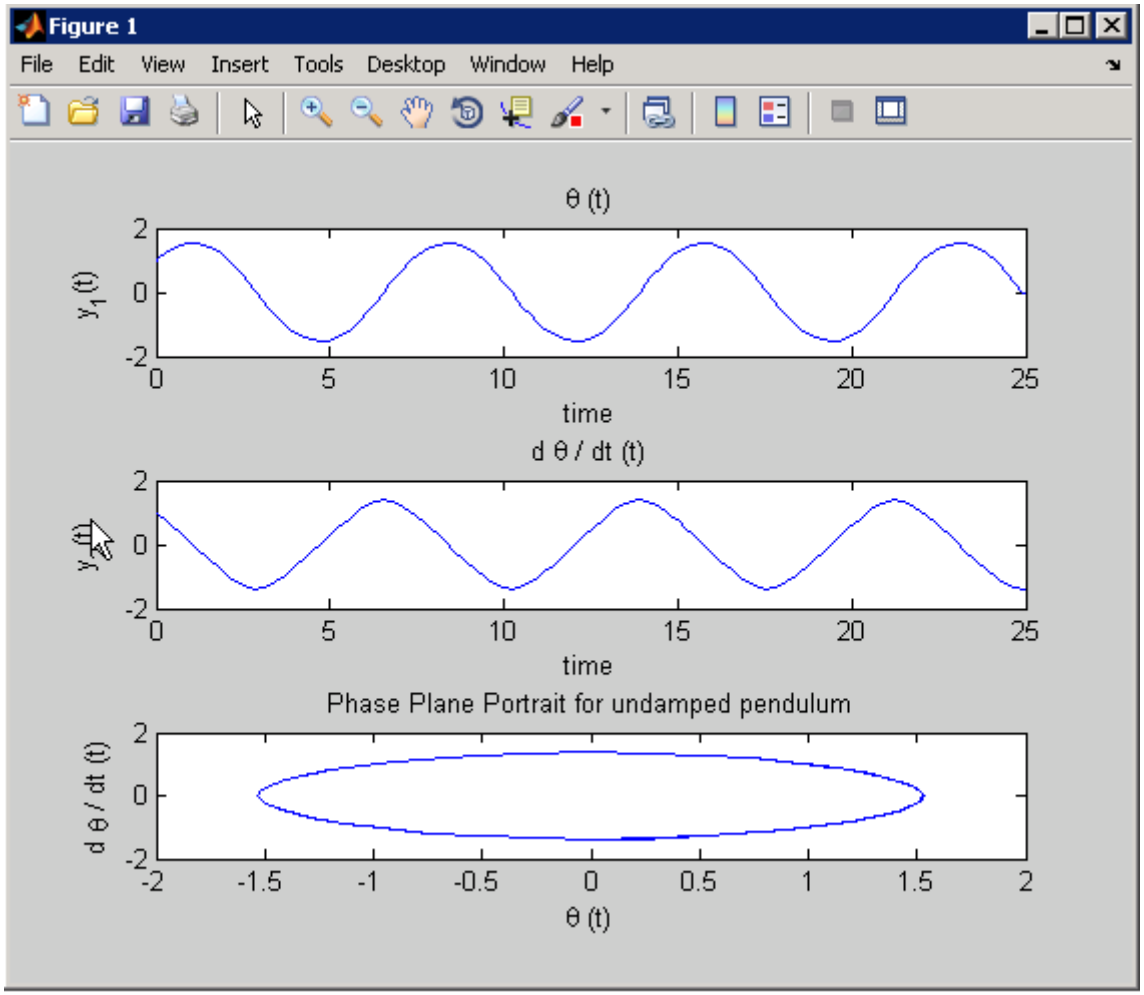

```
%now we can solve the ODE
%the parameter 'pendulum' call the script "pendulum.m" saved
%in one of the MATLAB curent workspace path
[t,y] = ode45('pendulum',[0 25], [1.0 1.0]);
subplot(3,1,1)
plot(t,y(:,1),'-');
xlabel('time');
ylabel('y_{1}(t)');
title('\theta (t)');
subplot(3,1,2)
plot(t,y(:,2),'-');
xlabel('time');
ylabel('y_{2}(t)');
title('d \theta / dt (t)');
subplot(3,1,3)
plot(y(:,1),y(:,2),'-');
xlabel('\theta (t)');
ylabel('d \theta / dt (t)');
title('Phase Plane Portrait for undamped pendulum');
```

Here is a resume of different MATLAB ODE solver:

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

Table 3 ODE solvers

The result is visible on next page:


Figure 1
File Edit View Insert Tools Desktop Window Help
θ (t)
y1(t)
time
d θ / dt (t)
y2(t)
time
Phase Plane Portrait for undamped pendulum
d θ / dt (t)
θ (t)

SAM

Lotka-Volterra (system of ODE)

Various assumptions during our study of Population Dynamics took us to the following system of ODE:

$$\frac{dN}{dt} = rN - cNP$$
$$\frac{dP}{dt} = -mP + bNP$$

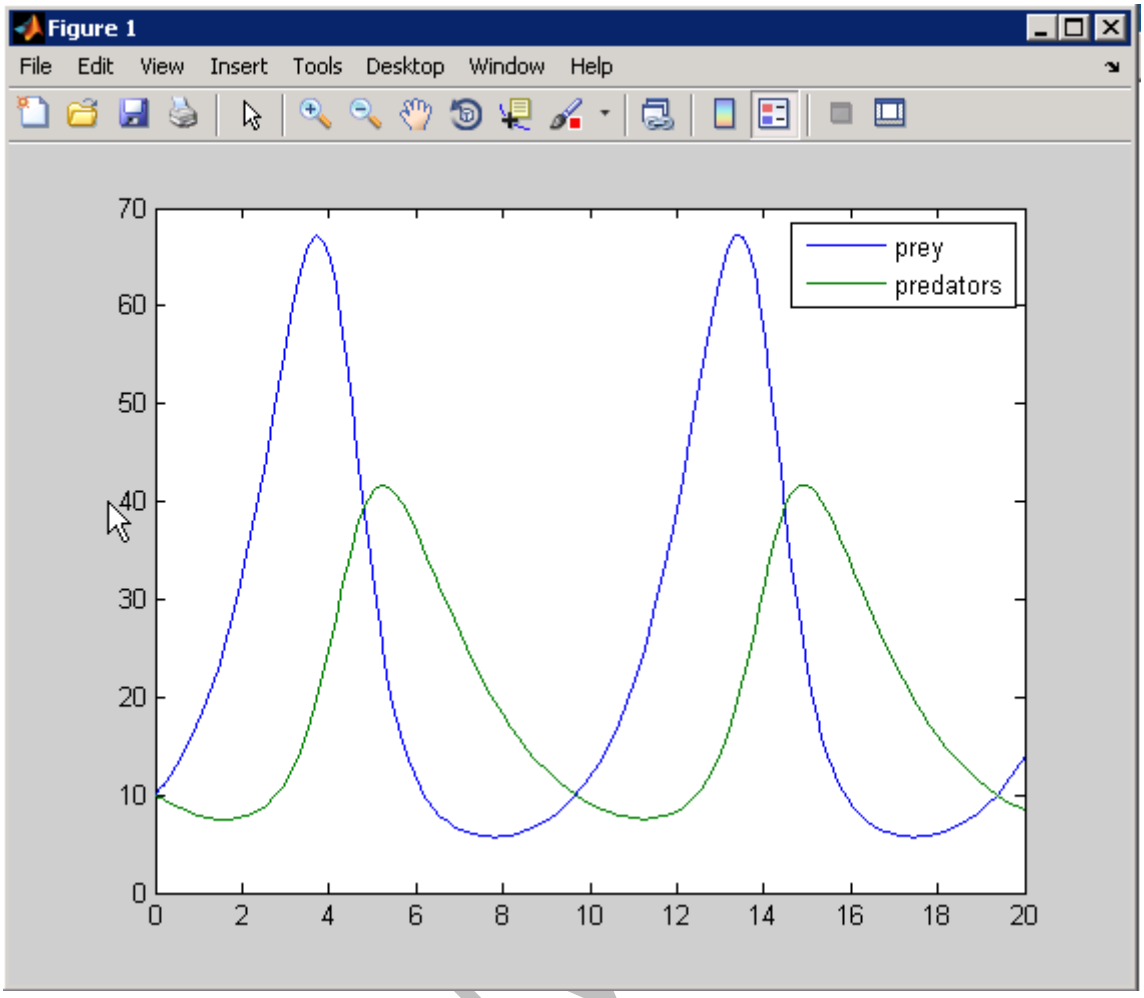
This will be rewritten in MATLAB into the following form for some given values of the constants of the system:

```
1   function dx = lotka_volterra(t, x)
2       dx = zeros(2,1);
3       alpha = 1;
4       beta = .05;
5       delta = .02;
6       gamma = .5;
7
8       dx(1) = alpha * x(1) - beta * x(1) * x(2);
9       dx(2) = delta * x(1) * x(2) - gamma * x(2);
10  end
```

Now solving this using once again **ode45( )** solver but also the function **odeset( )** to define the Relative Error Tolerance and to say that both variable are not negatives:

```
1   % We first set our preferences for ode45
2   % The default relative tolerance is 1e-3.
3   % To set our output to non-negative (population can not be negative),
4   % we provide an array listing each population that we want to constrain.
5   % Since this example has two populations, we pass the array [1 2]
6   options = odeset('RelTol', 1e-4, 'NonNegative', [1 2]);
7   [t,x] = ode45('lotka_volterra', [0 20], [10 10], options);
8   plot(t,x);
9   legend('prey', 'predators');
```

So we get:


Figure 1
File Edit View Insert Tools Desktop Window Help
prey
predators
0 10 20 30 40 50 60 70
0 2 4 6 8 10 12 14 16 18 20

SAM

Lorenz attractor (system of ODE)

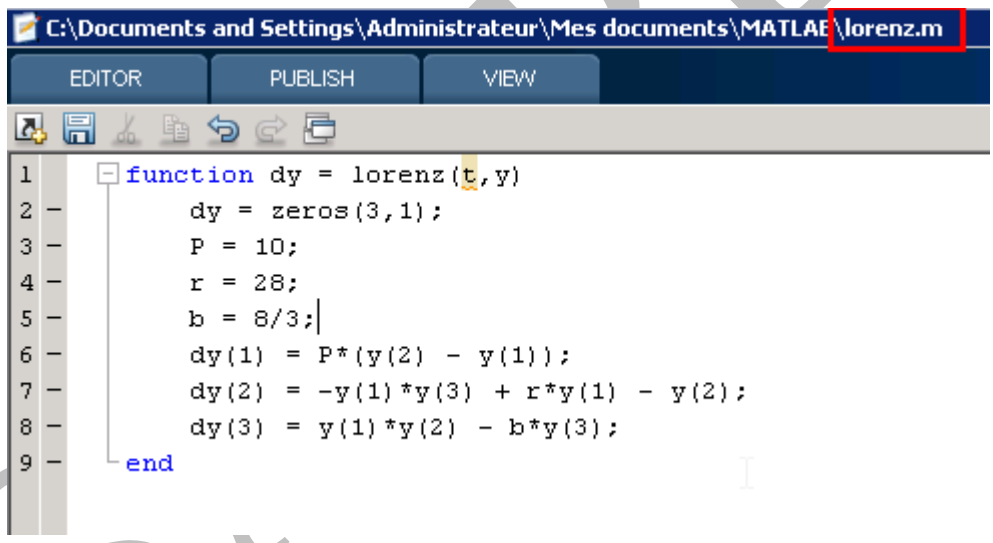
After many developments we get in the theoretical course of Meteorology Engineering the following system of ODE for the convection of a fluid:

$$\frac{dv}{dt} = -\Gamma v + \varepsilon g \frac{T_3}{2}$$
$$\frac{dT_3}{dt} = -KT_3 + \frac{vT_1}{r} - \frac{vT_4}{r}$$
$$\frac{dT_4}{dt} \cong -KT_4 + \frac{vT_3}{r}$$

And we get after some simplifications:

$$X' = \Pr(Y - X)$$
$$Y' = rX - Y - XZ$$
$$Z' = -Z + YX$$

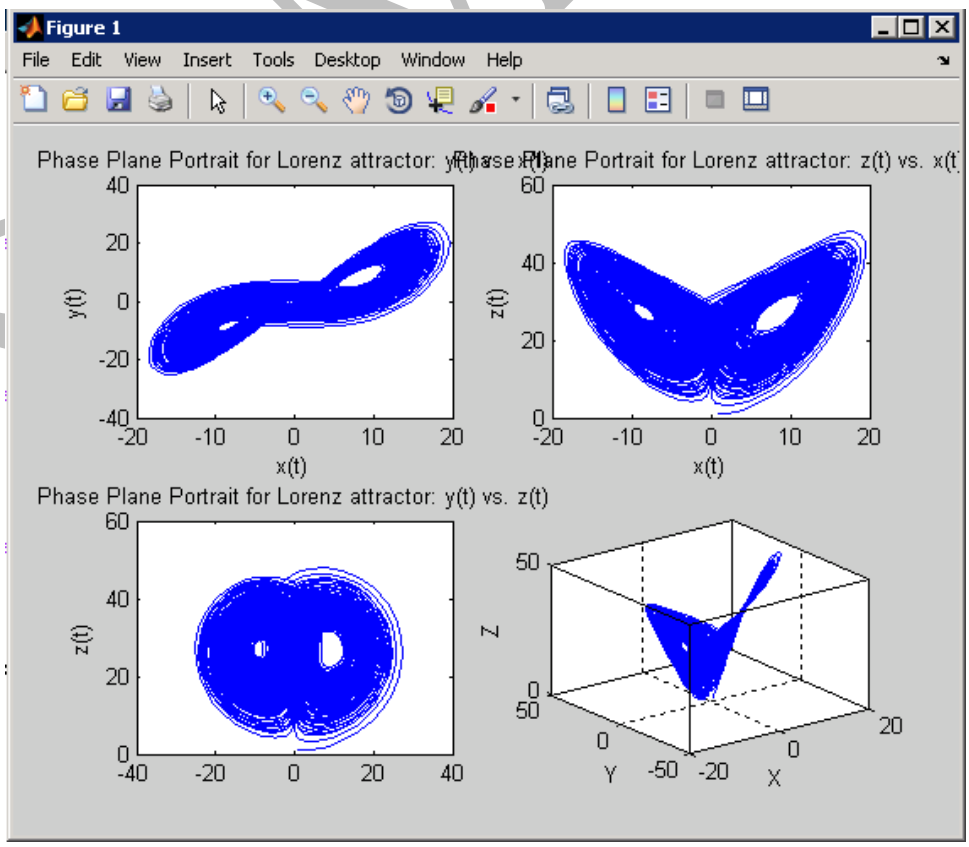
We will write this in MATLAB into the following way:



And after we solve it again using **ode45( )**:

```
[t,y] = ode45('lorenz',[0 250],[1.0 1.0 1.0]);
subplot(221)
plot(y(:,1),y(:,2),'-');
xlabel('x(t)');
ylabel('y(t)');
title('Phase Plane Portrait for Lorenz attractor: y(t) vs. x(t)');
subplot(222)
plot(y(:,1),y(:,3),'-');
xlabel('x(t)');
ylabel('z(t)');
title('Phase Plane Portrait for Lorenz attractor: z(t) vs. x(t)');
subplot(223)
plot(y(:,2),y(:,3),'-');
ylabel('y(t)');
ylabel('z(t)');
title('Phase Plane Portrait for Lorenz attractor: y(t) vs. z(t)');
subplot(224)
plot3(y(:,1),y(:,2),y(:,3))
grid on;box on;
xlabel('X');ylabel('Y');zlabel('Z');
```

We get the following famous plots:



2 body motion (system of ODE)

Consider 2 point masses, with masses m_1 and m_2, described by the two-dimensional coordinate positions:

$$\left(x_1(t), y_1(t)\right)$$
$$\left(x_2(t), y_2(t)\right)$$

The gravitational force between both of these masses is given by:

$$\frac{Gm_1m_2}{r_{12}^2} = \frac{Gm_1m_2}{\left(x_1 - x_2\right)^2 + \left(y_1 - y_2\right)^2}$$

If we apply Newton's 2nd law for both masses in each coordinate direction (*x* and *y*), we get a set of 4 coupled differential equations:

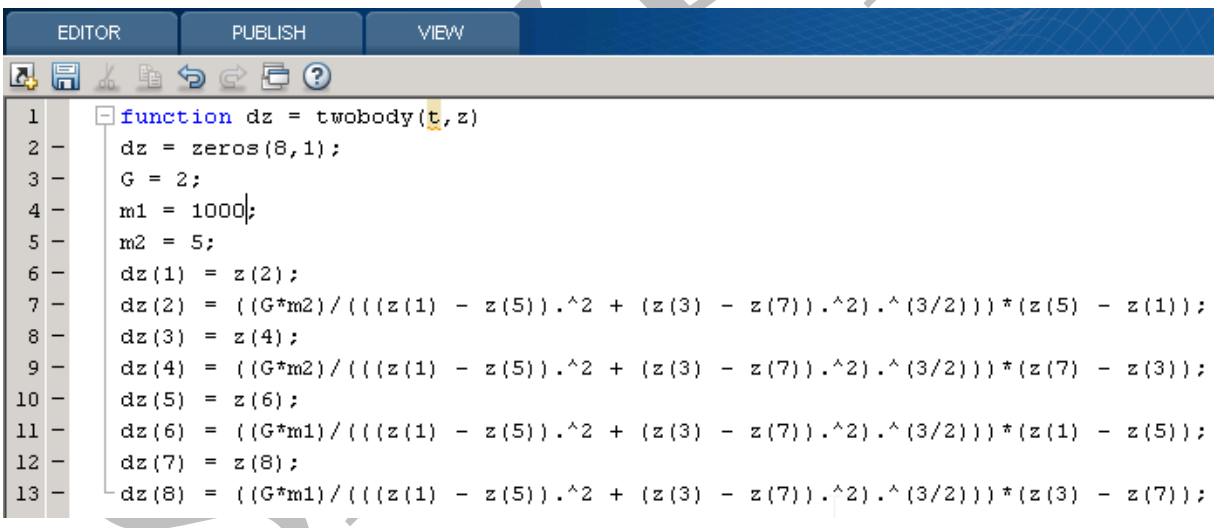
$$m_1 \frac{dx_1^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}\left(x_2 - x_1\right)$$
$$m_1 \frac{dy_1^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}\left(y_2 - y_1\right)$$
$$m_2 \frac{dx_2^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}\left(x_1 - x_2\right)$$
$$m_2 \frac{dy_2^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}\left(y_1 - y_2\right)$$

To solve these equations in MATLAB, we'll need to rewrite each of the 4 equations into two first order differential equations, thereby giving us a system of 8 differential equations to solve:

SAMPLE

$$\frac{dx_1}{dt} = u_1$$
$$\frac{du_1}{dt} = \frac{Gm_2}{r_{12}^3}(x_2 - x_1)$$
$$\frac{dy_1}{dt} = v_1$$
$$\frac{dv_1}{dt} = \frac{Gm_2}{r_{12}^3}(y_2 - y_1)$$
$$\frac{dx_2}{dt} = u_2$$
$$\frac{du_2}{dt} = \frac{Gm_1}{r_{12}^3}(x_1 - x_2)$$
$$\frac{dy_2}{dt} = v_2$$
$$\frac{dv_2}{dt} = \frac{Gm_1}{r_{12}^3}(y_1 - y_2)$$

In MATLAB this gives:




```
1   function dz = twobody(t,z)
2   dz = zeros(8,1);
3   G = 2;
4   m1 = 1000;
5   m2 = 5;
6   dz(1) = z(2);
7   dz(2) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(5) - z(1));
8   dz(3) = z(4);
9   dz(4) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(7) - z(3));
10  dz(5) = z(6);
11  dz(6) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(1) - z(5));
12  dz(7) = z(8);
13  dz(8) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(3) - z(7));
```

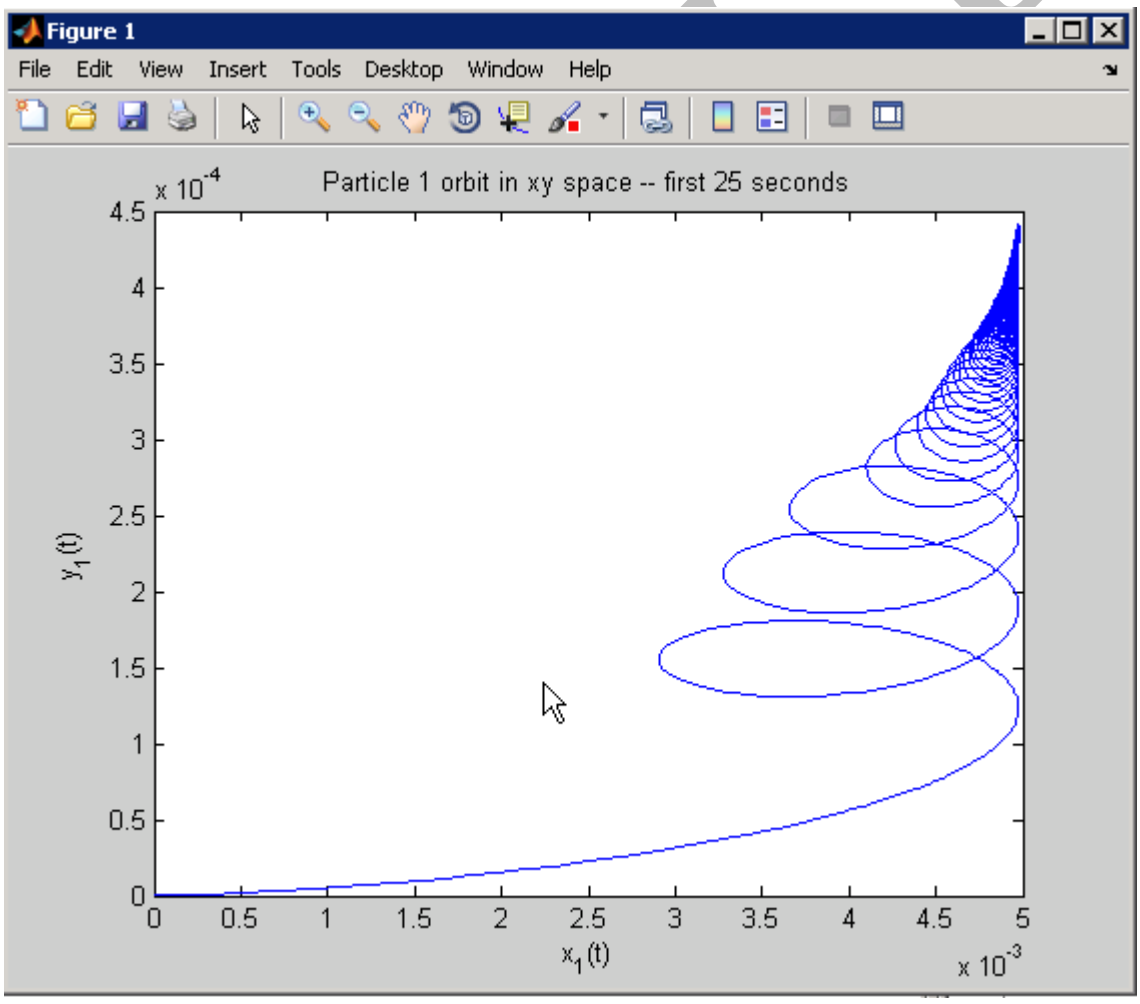
And now we solve again with ode45( ) between times [0, 25] and for initial values:

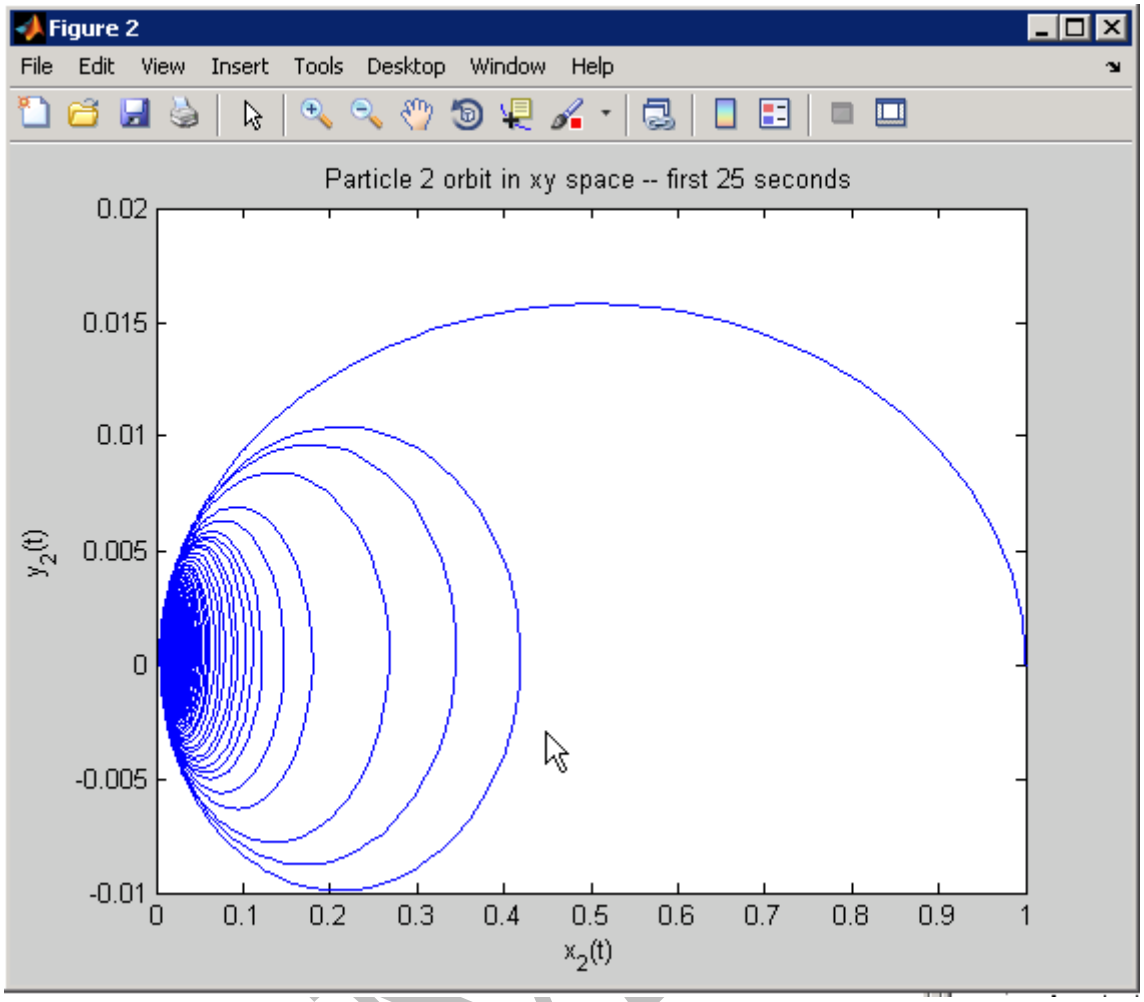
$$x_1(0), u_1(0), y_1(0), v_1(0), x_2(0), u_2(0), y_2(0), v_2(0)$$

EDITOR PUBLISH VIEW

```
1 -    clear all;close all;clc
2 -    [t,z] = ode45('twobody',[0 25], [0 0 0 0 1 0 0 1]);
3 -    plot(z(:,1),z(:,3),'-');
4 -    xlabel('x_{1}(t)');
5 -    ylabel('y_{1}(t)');
6 -    title('Particle 1 orbit in xy space -- first 25 seconds');
7 -    figure;
8 -    plot(z(:,5),z(:,7),'-');
9 -    xlabel('x_{2}(t)');
10 -   ylabel('y_{2}(t)');
11 -   title('Particle 2 orbit in xy space -- first 25 seconds');
```

And as output we get the following charts:




Figure 2
File Edit View Insert Tools Desktop Window Help
Particle 2 orbit in xy space -- first 25 seconds
0.02
0.015
0.01
0.005
0
-0.005
-0.01
$y_2(t)$
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$x_2(t)$

SAM

Heat equation (parapolic PDE)

Working with PDE in MATLAB is like in Maple... something horrible to write and time consuming to search the right syntax.

Let us begin with the famous heat equation that we have studied and proved in the course of Thermodynamics:

$$\frac{\partial T(x,t)}{\partial t} - \frac{\kappa}{\rho c_p}\frac{\partial^2 T(x,t)}{\partial x^2} = 0$$

using the function **pdepe( )** (the explanations are given after):

```
EDITOR  PUBLISH  VIEW

1  function heat_equation()
2      m=0;
3      x=linspace(0,1,30); %this creates 30 equal points along the length of the rod
4      t=linspace(0,15,20); %this creates 20 equal time points
5      sol=pdepe(m,@pdex,@pdexic,@pdexbc,x,t);
6      u=sol(:,:,1);
7      surf(x,t,u);xlabel('Position');ylabel('time');zlabel('Temperature');
8      figure;
9      plot(x,u);xlabel('length');ylabel('Temperature');
10 end
11
12 function [c,f,s] = pdex(x,t,u,DuDx)
13     c = 1;
14     f = 0.02*DuDx;
15     s = 0;
16 end
17
18 function u0 = pdexic(x)
19     u0 = 50;  %initial temperature
20 end
21
22 function [pl,ql,pr,qr] = pdexbc(xl,ul,xr,ur,t)
23     pl = ul;
24     ql = 0;
25     pr= ur-100;
26     qr = 0;
27 end
```

For **pdepe( )** to be able to interpret the PDE the equation must in MATLAB written into the general form of:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

where *c* is a diagonal matrix, *f* is called the "flux term", and *s* is called the "source term". Placing our PDE in the form as shown above, starting from (using the coefficient of the above example):

$$\frac{\partial u}{\partial t} = 0.02\frac{\partial^2 u}{\partial x^2}$$

it becomes:

$$1\cdot\left(\frac{\partial u}{\partial t}\right) = \left(0.02\frac{\partial}{\partial x}\right)\frac{\partial u}{\partial x} + 0$$

Therefore:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right) = 1$$
$$f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0.02\frac{\partial u}{\partial x}$$
$$m = 0$$
$$s\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0$$

This is why we have in *pdex*:

```
function [c,f,s] = pdex(x,t,u,DuDx)
    c = 1;
    f = 0.02*DuDx;
    s = 0;
end
```

Next is **pdexic( )**. This in the function that evaluates the initial conditions of the PDE. Since the initial temperature distribution for this example is:

$$u_0(x) = 50$$

the function has the following form:

```
function u0 = pdexic(x)
    u0 = 50;  %initial temperature
end
```

If we were to write an initial condition function for a piecewise (triangular) function, the function would be written as follows:

```
function u0=pdexic(x)
    If x<1
        u0=x;
    Else
        u0=1-x;
end
```

SAMPLE

The function **pdexbc( )** is the function that evaluates the boundary conditions. Again as for **pdepe( )** to be able to evaluate these conditions they must be put into a form that it can be interpreted by MATLAB. The general form for the boundary conditions is:

$$p(x,t,u)+q(x,t)\,f\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

With the boundary conditions of our example being:

$$u(0,t)=ul=0 \quad \text{and} \quad u(1,t)=ur=100$$

We need to find the values for *pl* and *ql*. Where *pl* and *ql* are the functions in the general form for **pdepe( )** of the left boundary conditions:

$$u(0,t)+0\cdot\frac{\partial u}{\partial x}(0,t)=0$$

This gives:

$$pl=ul$$

Where *ul* is the temperature function at the left boundary, and:

$$ql=0$$

Next we find the **pdepe( )** form for *pr* and *qr*, where these are the values for the right boundary condition:

$$u(1,t)+0\cdot\frac{\partial u}{\partial x}(1,t)=100$$

Therefore:

$$pr=ur-100$$

And:

$$qr=0$$

This function is written as follows:

```
function [pl,ql,pr,qr] = pdexbc(xl,ul,xr,ur,t)
    pl = ul;
    ql = 0;
    pr= ur-100;
    qr = 0;
end
```

For an example of the Newmann conditions, where:

$$u(0,t)=0 \quad \text{and} \quad u(1,t)=0$$

SAMPLE

are the boundary conditions. We find the **pdepe( )** form for *pl* and *ql*:

$$u(0,t)+1\cdot\frac{\partial u}{\partial x}(0,t)=0$$

Therefore, we have :

$$pl=0, ql=0$$

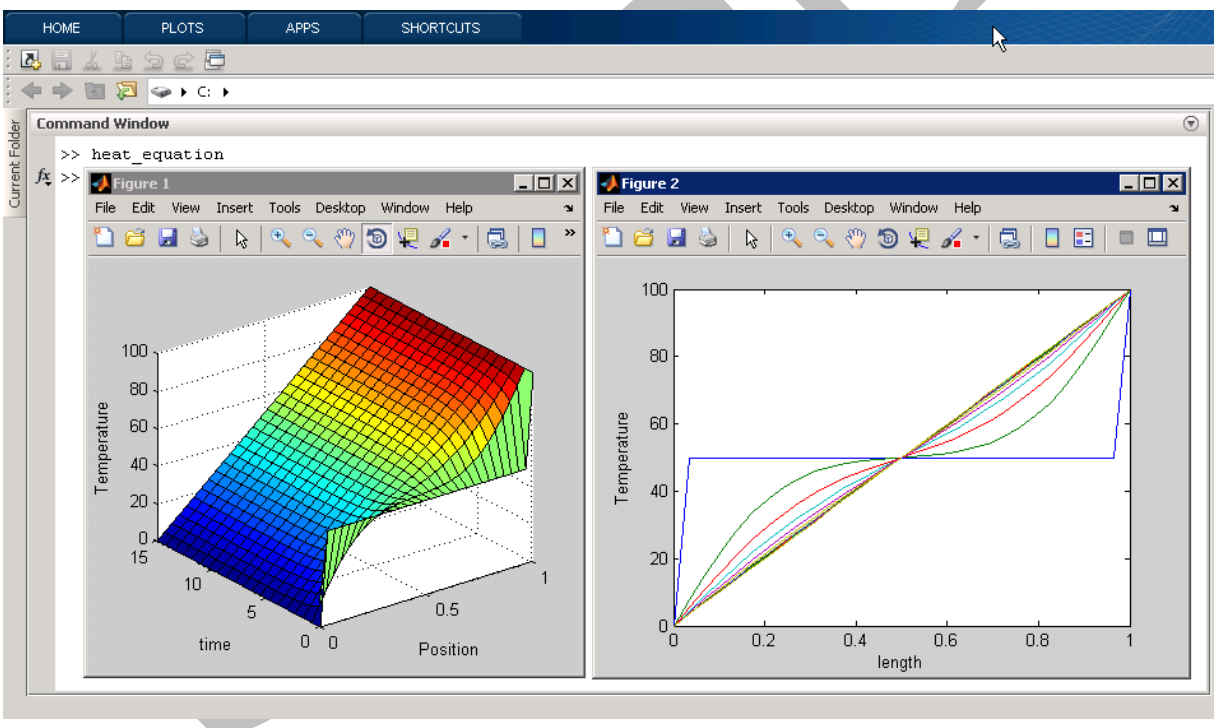
Evaluating the right boundary:

$$u(0,t)+1\cdot\frac{\partial u}{\partial x}(1,t)=0$$

gives:

$$pr=0, qr=0$$

The result we get the function *heat_equation* is:



Fast Fourier Transform (DFT)

The FFT[1] is used a lot in signal processing but also in finance to identify periodicity of a time series.

The function **fft( )** is not easy (intuitive) to implement to get what we saw in the theoretical course.

Note: We recommend the user the see a more real example on sound processing at page 537

Then see below how we can get the expected result:

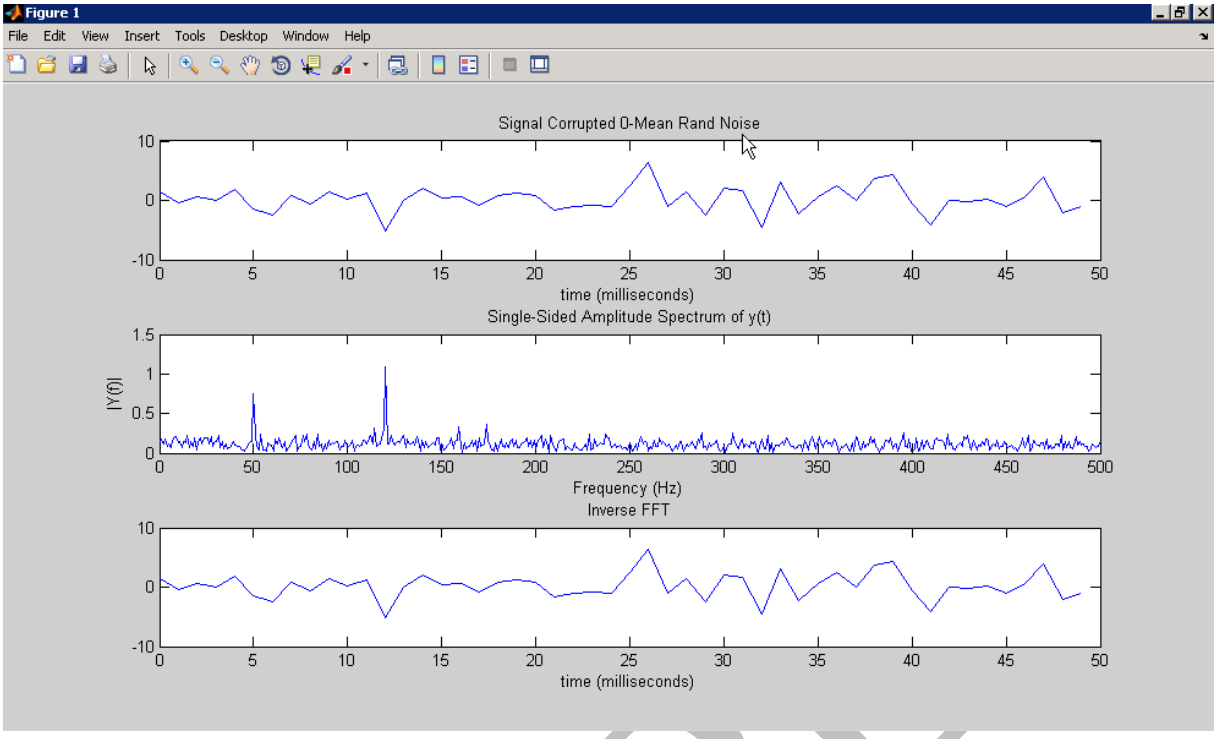
EDITOR PUBLISH VIEW

```
%Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and 120 Hz
%sinusoid of amplitude 1 and corrupt it with some zero-mean random noise:
Fs=1000;                    % Sampling frequency
T=1/Fs;                     % Sample time
L=1000;                     % Length of signal
t=(0:L-1)*T;                % Time vector
% Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x=0.7*sin(2*pi*50*t)+sin(2*pi*120*t);
y=x+2*randn(size(t));     % Sinusoids plus noise
%plot of signal
subplot(3,1,1);plot(Fs*t(1:50),y(1:50));
title('Signal Corrupted 0-Mean Rand Noise');xlabel('time (milliseconds)')
%FFT of signal
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);
% Plot single-sided amplitude spectrum.
subplot(3,1,2);plot(f,2*abs(Y(1:NFFT/2+1)));
title('Single-Sided Amplitude Spectrum of y(t)');
xlabel('Frequency (Hz)');ylabel('|Y(f)|');
%The main reason the amplitudes are not exactly at 0.7 and 1 is
%because of the noise.

%inverse FFT
y=ifft(Y)*L;
subplot(3,1,3);
plot(Fs*t(1:50),y(1:50))
title('Inverse FFT');xlabel('time (milliseconds)')
```

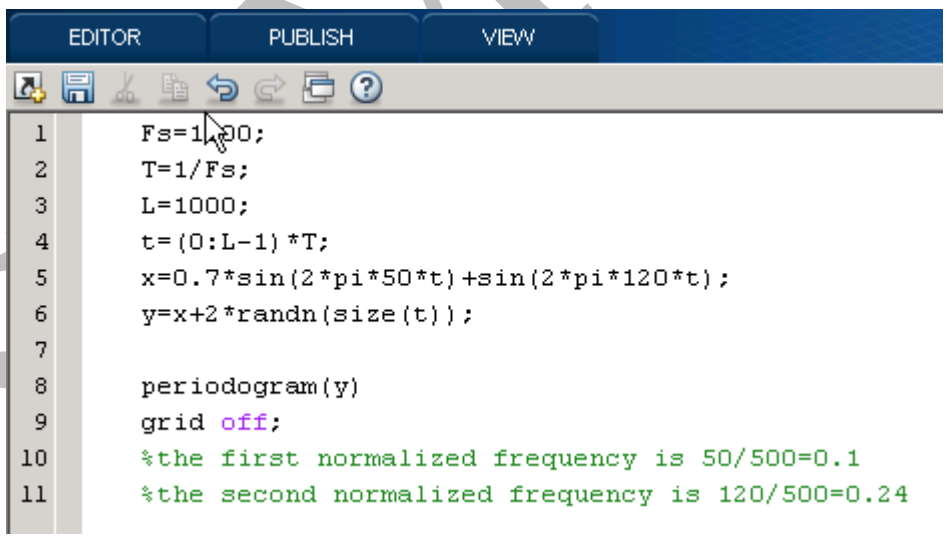
In the second subplot you have the frequencies **and** the amplitudes of the original signal!

[1] FFT is the name for *any* efficient algorithm that can compute the DFT in about $\Theta(n \log n)$ time, instead of $\Theta(n^2)$ time. There are several existing FFT algorithms.



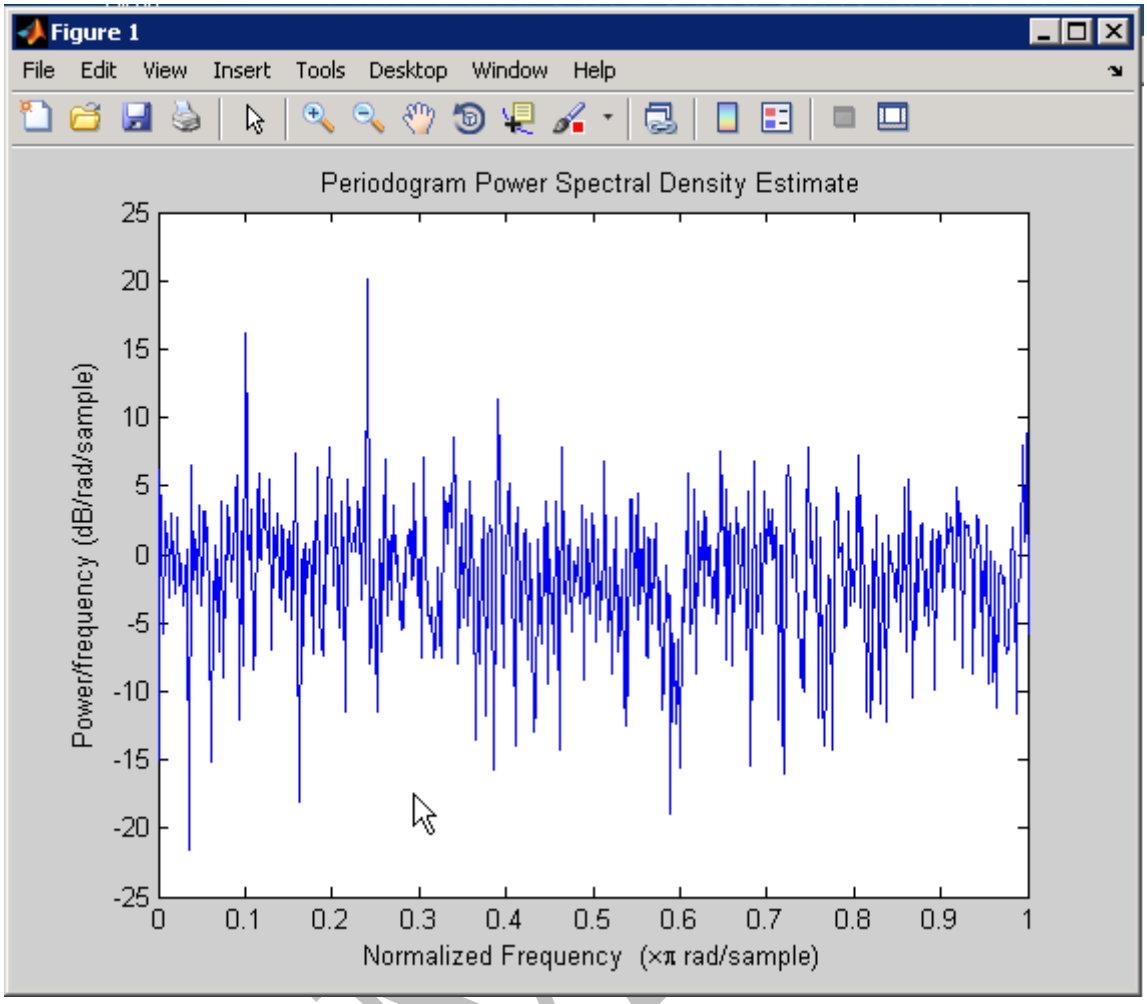
Note: We will see during our study of the Signal Processing (see page 1135) Toolbox how to quickly determined peaks positions and plot them.

Some people prefer to use the periodogram but this command is reserved to the Signal Processing Toolbox:



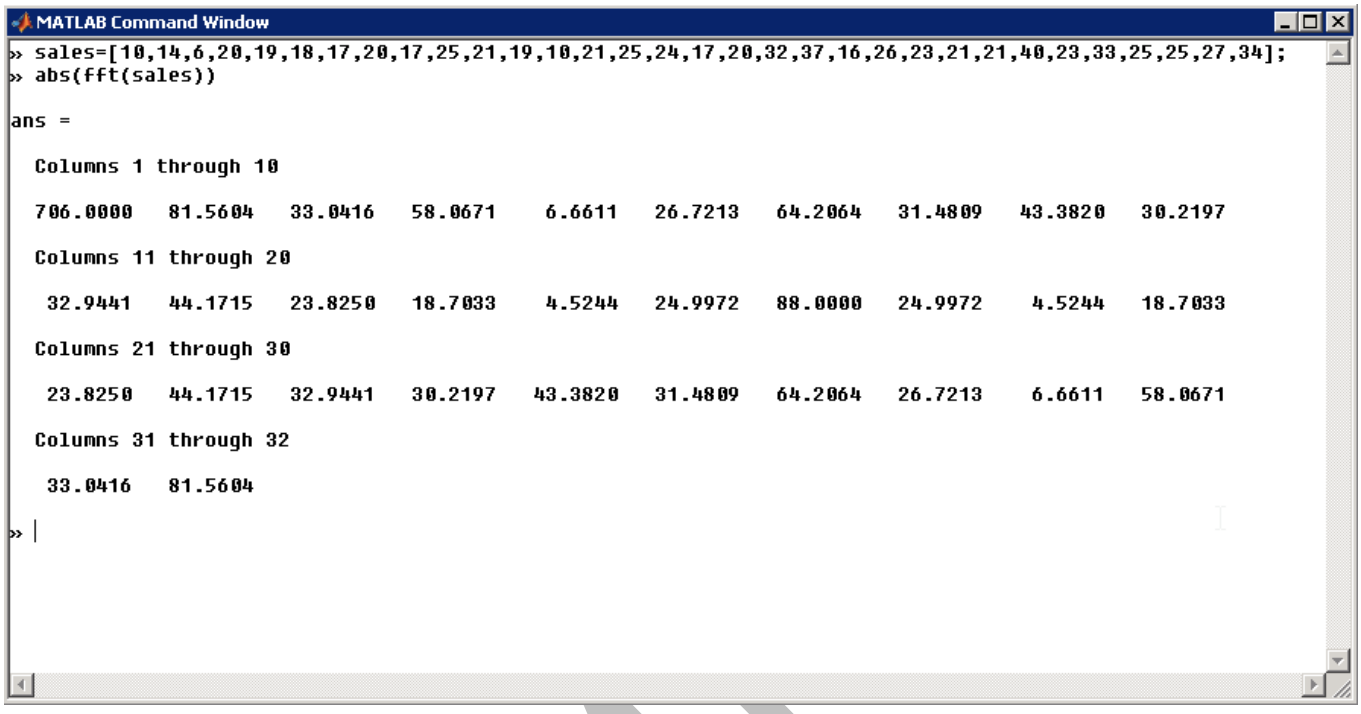

```
1     Fs=1000;
2     T=1/Fs;
3     L=1000;
4     t=(0:L-1)*T;
5     x=0.7*sin(2*pi*50*t)+sin(2*pi*120*t);
6     y=x+2*randn(size(t));
7
8     periodogram(y)
9     grid off;
10    %the first normalized frequency is 50/500=0.1
11    %the second normalized frequency is 120/500=0.24
```

The result is:


Figure 1
File Edit View Insert Tools Desktop Window Help
Periodogram Power Spectral Density Estimate
Power/frequency (dB/rad/sample)
Normalized Frequency (×π rad/sample)
25
20
15
10
5
0
-5
-10
-15
-20
-25
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

SAMPLE

And now have a look if for the statistical test of periodicity that we have studied in the theoretical training we get the same coefficients (module of complex numbers) as for R and Microsoft Excel:



So we get exactly the expected values!

SAMPLE

Image processing

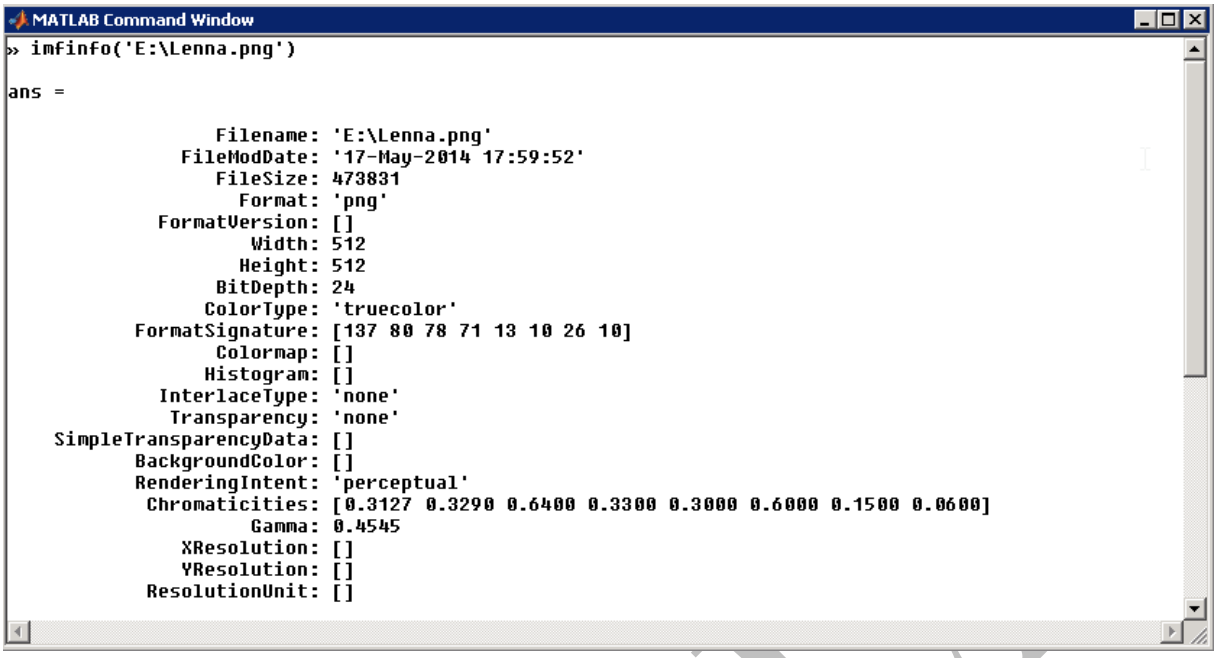
Without the Image Processing Toolbox working with images in MATLAB is possible but requires a lot of manipulations of matrices to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without too much pain using native core function.

Get image properties

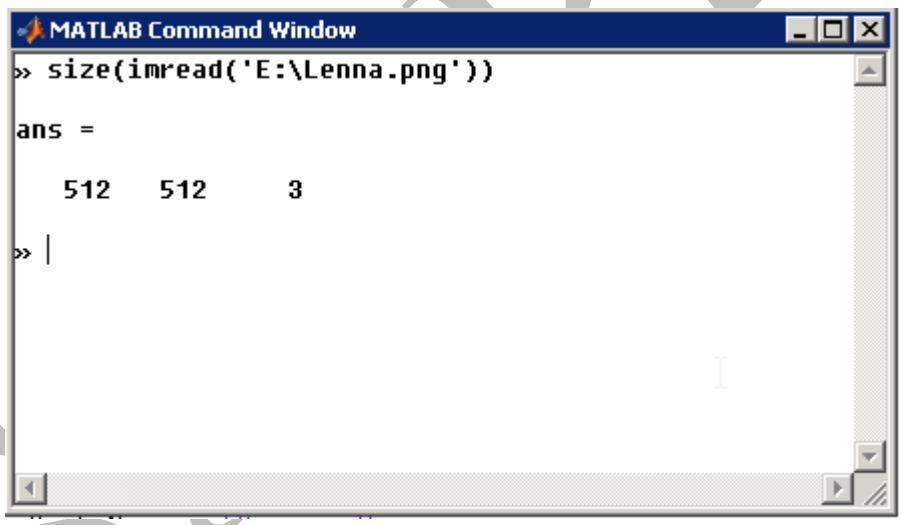
The basics of working with images are to retrieve information concerning the image file itself. Consider the following image:



Using the function **iminfo( )** we get:



The most important property for a lot of images effects and manipulations is the size of the image that we can simple get using the function **size( )**:



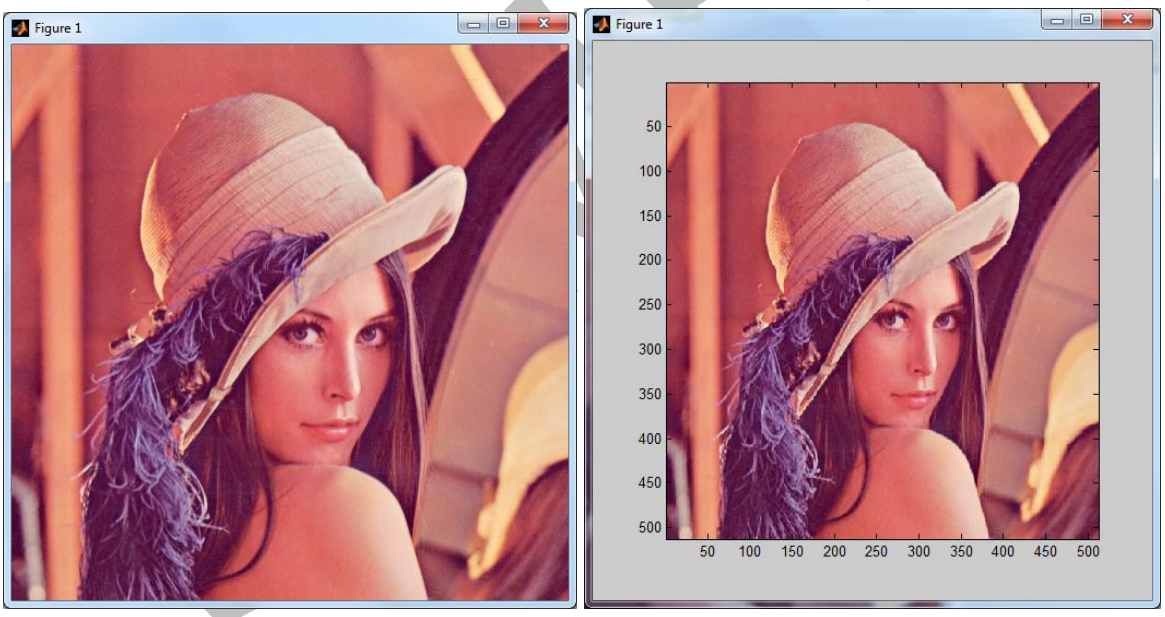
Show an image properly

Show properly and image is important for the end user. For this purpose consider the following script:

```
1   img=imread('Lenna.png');
2
3   figure;
4   imshow(img); %
5   set(gcf,'menubar','none');
6
7   figure
8   image(img);
9   set(gcf,'menubar','none');
```

Caution: **imshow( )** belongs to the Image Processing Toolbox.

With this we get for the first figure (left) on the second figure (right):

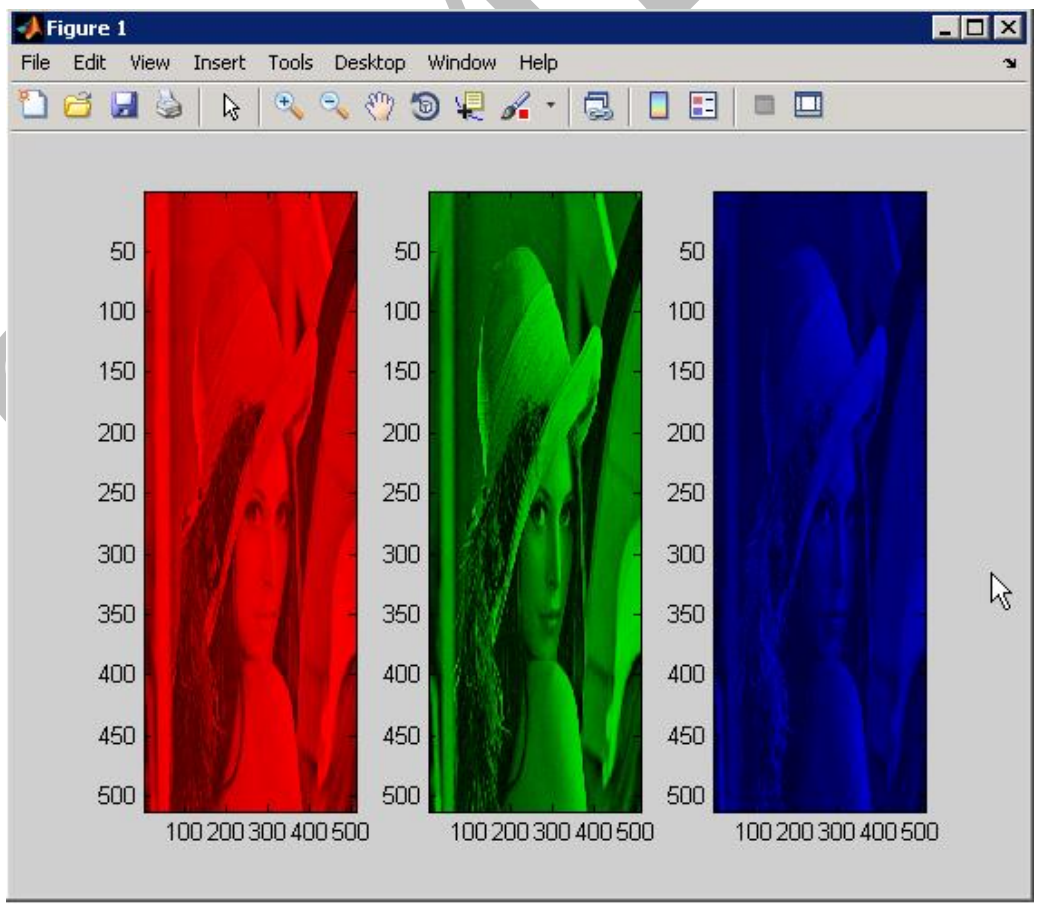


Decompose images alpha layers

This is a very good exercise to understand how image files are builded on the basis of three matrices containing respectively the RGB color channels:

```
1   img=imread('E:\Lenna.png');
2   imgrouge=uint8(zeros(size(img)));
3   imgrouge(:,:,1) = img(:,:,1);
4   subplot(1,3,1);
5   image(imgrouge)
6   imgvert= uint8(zeros(size(img)));
7   imgvert(:,:,2)=img(:,:,2);
8   subplot(1,3,2);
9   image(imgvert)
10  imgbleu=uint8(zeros(size(img)));
11  imgbleu(:,:,3)=img(:,:,3);
12  subplot(1,3,3);
13  image(imgbleu);
```

The result will be:

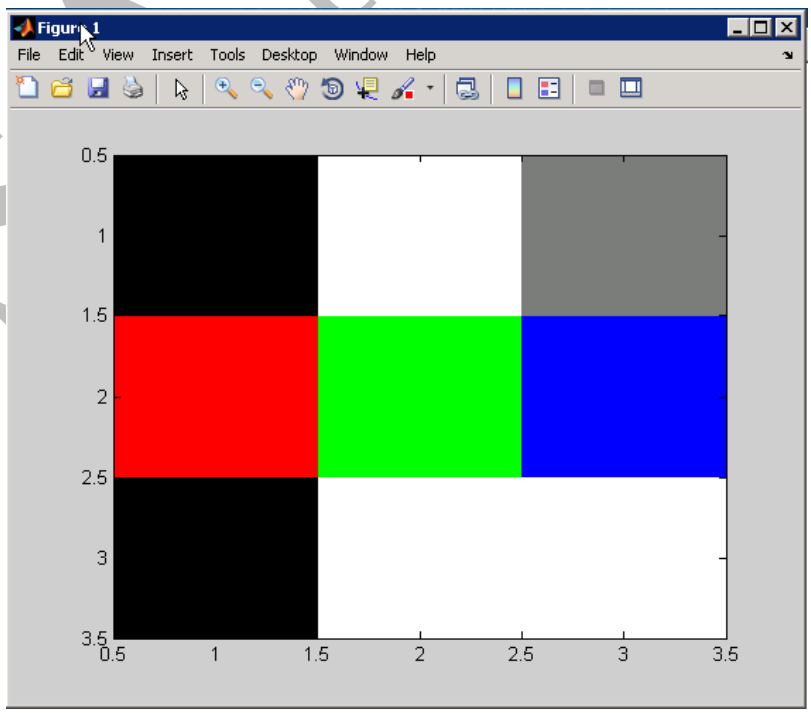


Build an image

To understand how a three layer image works a very good exercise is to build its own RGB image and also to see how MATLAB behaves in some extreme cases:

```
1   clear all;close all;clc
2
3   img=uint8(zeros(3,3,3));
4
5   %first row
6   img(1,1,:)=0; %pure black
7   img(1,2,:)=255; %pure white
8   img(1,3,:)=125; %pure gray
9
10  %second row
11  img(2,1,1)=255; %pure red
12  img(2,2,2)=255; %pure green
13  img(2,3,3)=255; %pure blue
14
15  %third row: extrem cases to see the behavior
16  img(3,1,:)=-30;
17  img(3,2,:)=270;
18  img(3,3,:)=Inf;
19
20  image(img);
```

This will give:

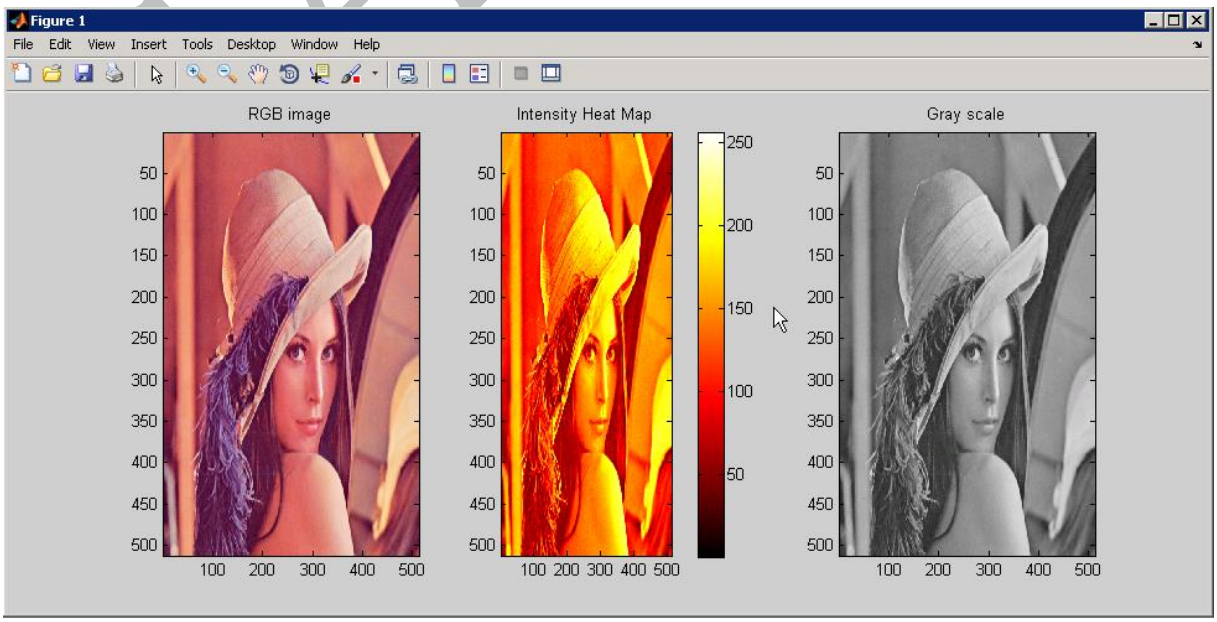


Plot some basic images manipulations and save image

We will see here below some basics images manipulations (show only some color channel and convert to gray without using Image Processing Toolbox) of colors and plots using the function **imread( )**, **image( )** and finally **imwrite( )** to save the transformed image on the computer:

```
1   subplot(1,3,1);
2   rgb = imread('E:\Lenna.png');
3   image(rgb) %image with full colors
4   title('RGB image')
5
6   subplot(1,3,2);
7   %image with average colors of the 3 alpha channel
8   im = mean(rgb,3);
9   image(im);
10  colorbar;
11  title('Intensity Heat Map');
12  colormap(hot(256));
13  imwrite(im,hot(256),'E:\LennaHeat.jpg','jpg');
14
15  subplot(1,3,3);
16  rgb=imread('E:\Lenna.png');
17  %image in gray scale
18  imgray=uint8(repmat(mean(rgb,3),[1,1,3]));
19  image(imgray)
```

The result of the multiple plot will be:

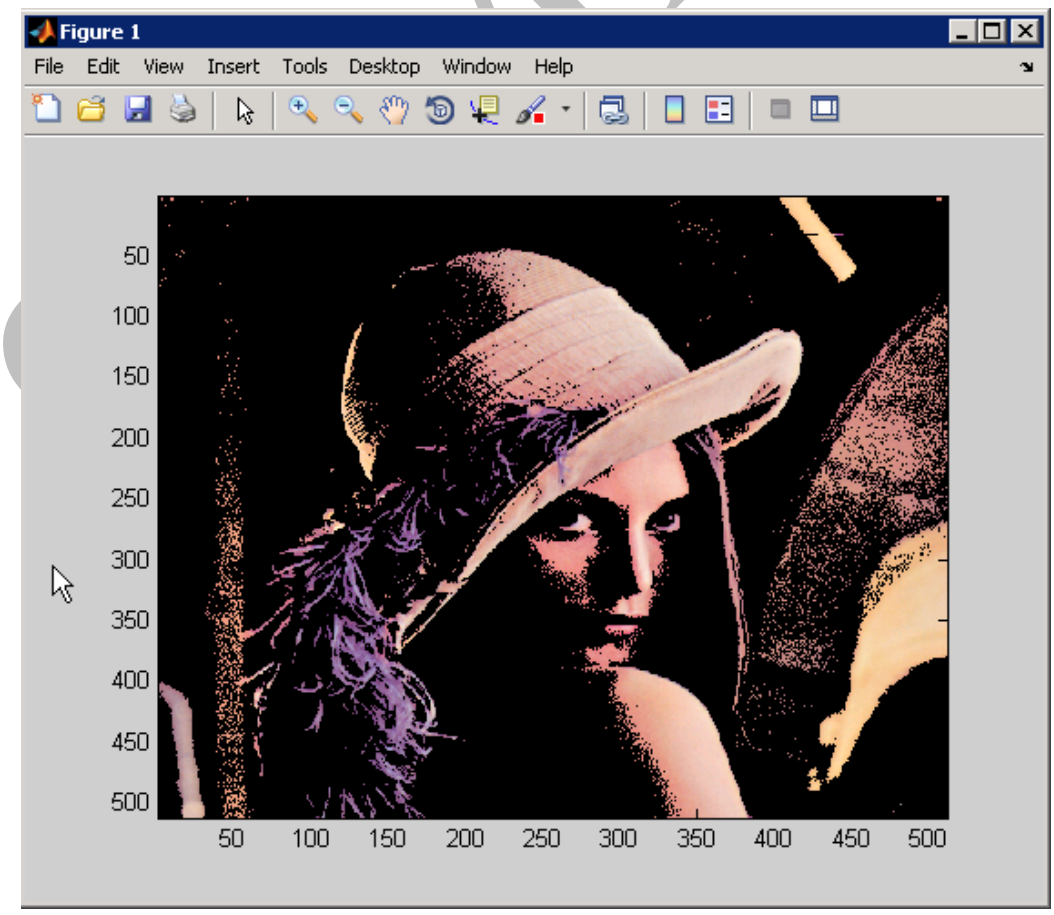


Filter on luminosity value of a given channel

This is also a very academic useful example to understand how MATLAB works with images. The example can be understood only by reading the script:

```
1  clear all;close all;clc
2  img=imread('Lenna.png');
3
4  s=size(img);
5
6  img_post=uint8(zeros(s));
7  treshold=125;
8
9  for k=1:s(1) %for each column
10
11     for l=1:s(2) %and for each row of the column
12         if img(k,l,3)>treshold %filter the third (blue layer)
13             img_post(k,l,:)=img(k,l,:); %keeep the pixel color info on the 3 layers
14         end
15     end
16
17 end
18
19 imshow(img_post)
```

The final result will be:



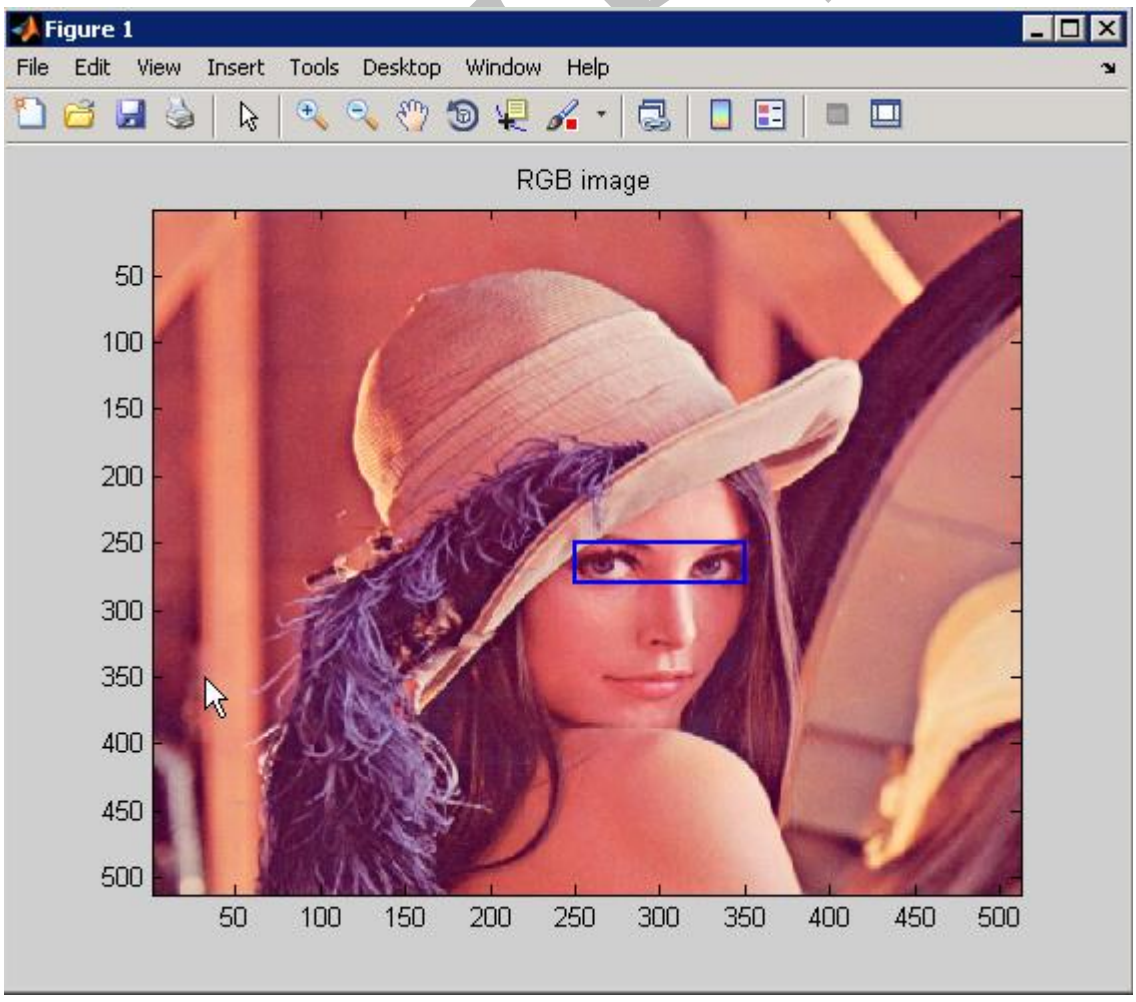
Draw on an image and export the result

Something very important in image pattern identification is to be able (later in this document) to recognize automatically geometries on a photo and to draw something around to put them in evidence for the user and after to save it on the computer.

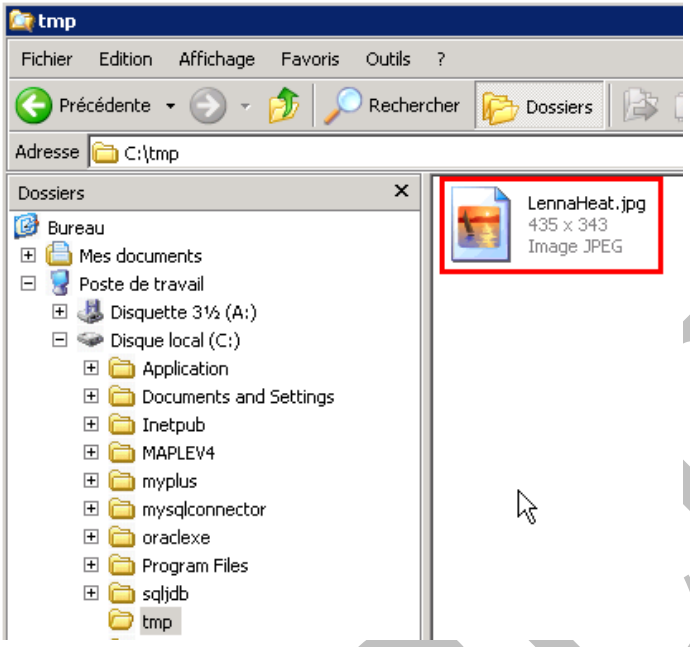
For this purpose we use the functions **rectangle( )**, **getframe( )** and **frame2im( )**:

```
rgb=imread('E:\Lenna.png');
image(rgb);
title('RGB image');
hold on;
rectangle('position',[250 250 100 30],'LineWidth',2, 'EdgeColor','b')
%and here we export the image
f=getframe(gca);
[rgb] = frame2im(f);
imwrite(rgb,'C:\tmp\LennaHeat.jpg','jpg');
```

Thus we get:



And it will create a file:



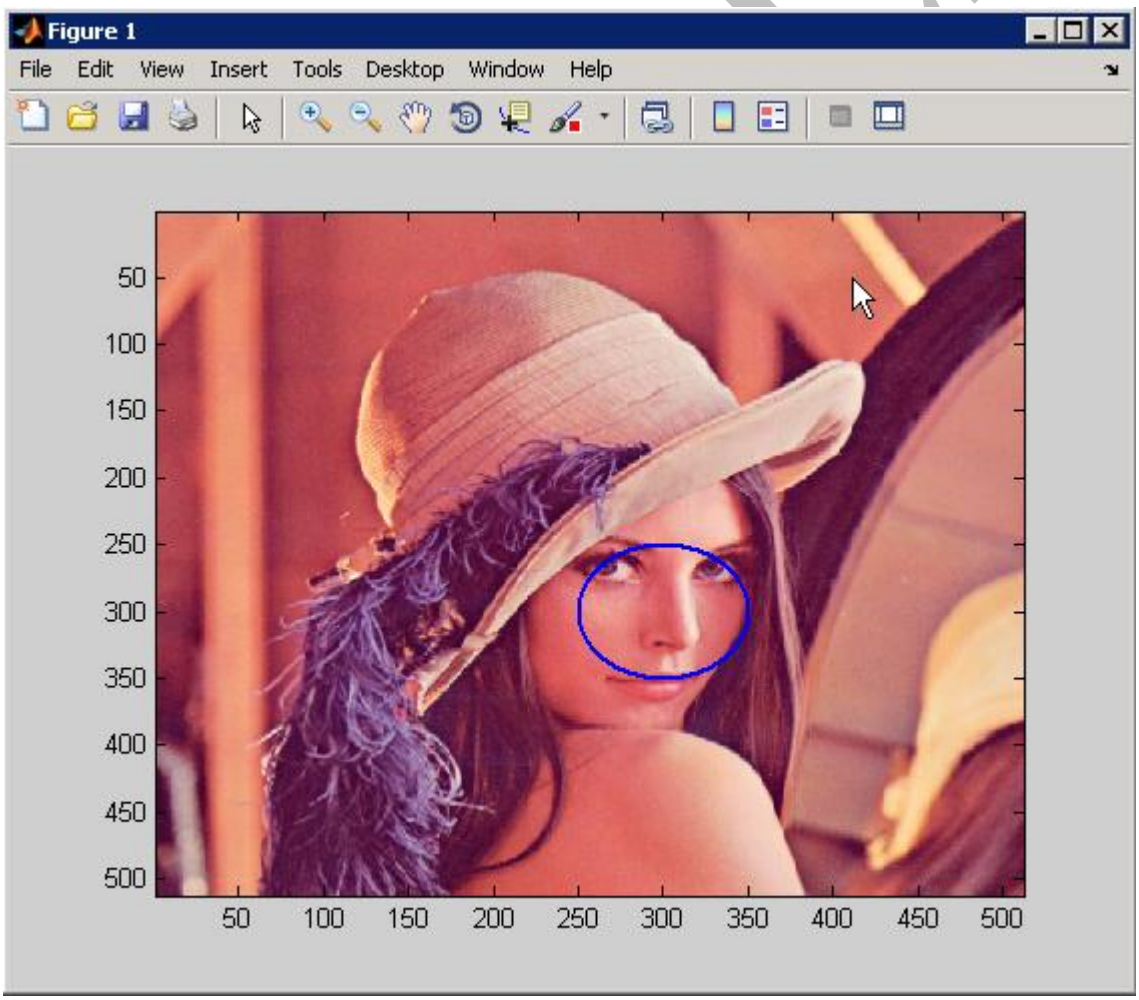
who's preview gives:



Or if you want to draw a circle or an ellipse:

```
1  -    clear all;close all;clc
2
3  -    img=imread('Lenna.png');
4  -    image(img);
5  -    rectangle('position',[250 250 100 100],'LineWidth',2,...
6           'EdgeColor','b','Curvature',[1 1]);
7
8  -    f=getframe(gca);
9  -    img=frame2im(f);
10 -    imwrite(img,'LennaReconnaissance.jpg');
```

That will give:

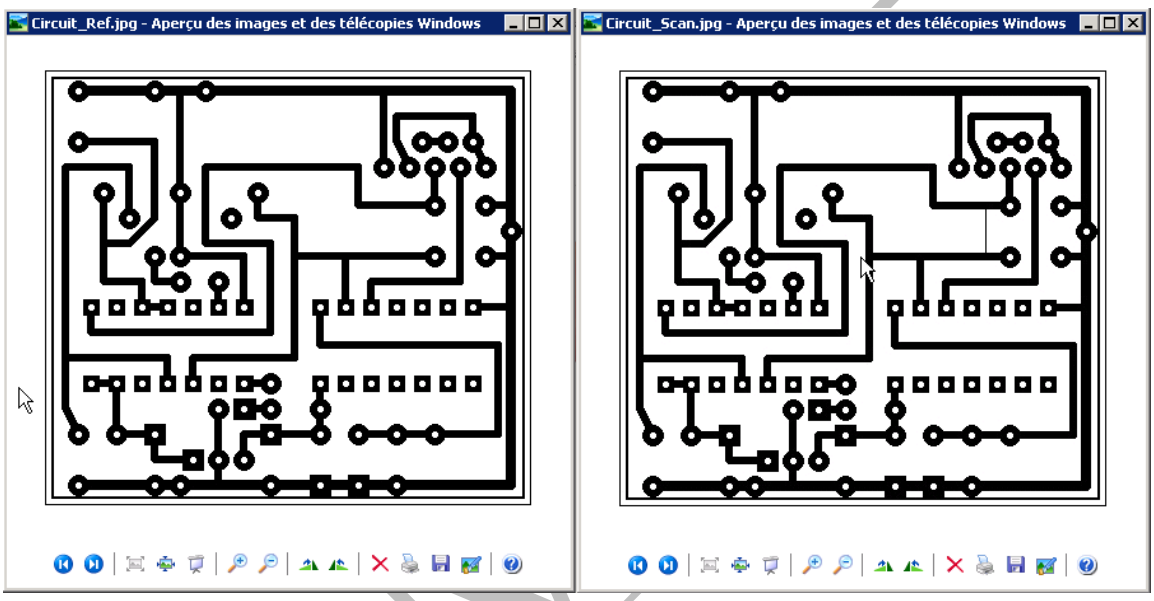


Compare images

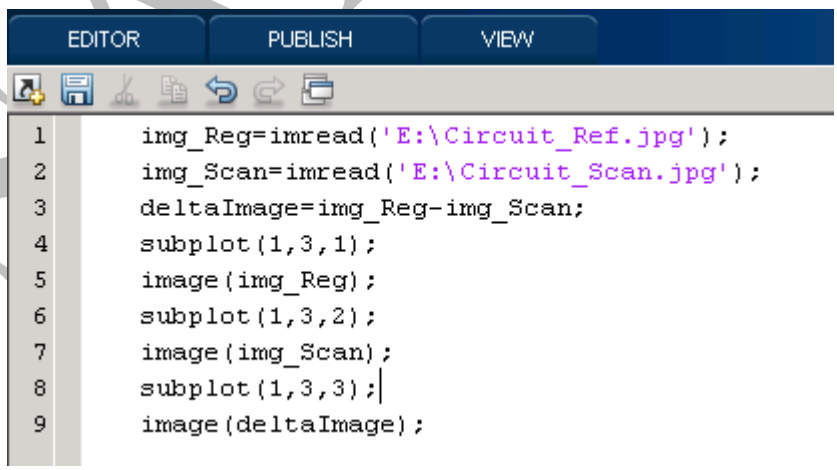
The comparison of images is a very important subject in quality control of production processes to compare pieces to a reference one and rejecting the too bad one.

Caution! The images (when you don't have the Image Processing Toolbox) must have the same size!

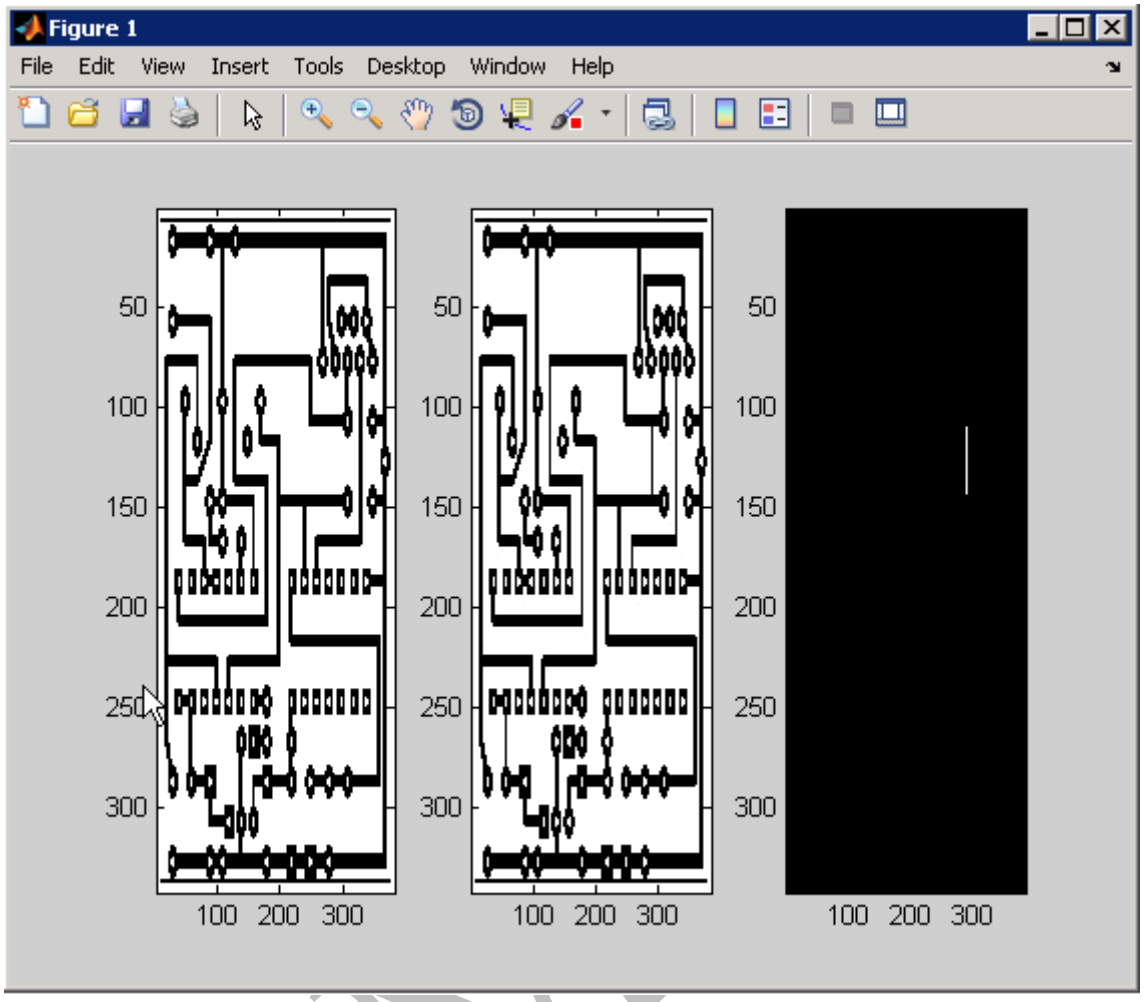
For the example consider the two images below. The reference is on the left, the tested one is on the right (for the conversion of an image in black-white see the chapter of Image Processing Toolbox on page 1180):



The comparison simply uses a subtraction of the matrices of the both images:



The plotted result is on the right. After we just have to make an arithmetic sum of all pixels to know if we have to significate difference or not:


Figure 1
File Edit View Insert Tools Desktop Window Help
50
100
150
200
250
300
100 200 300
50
100
150
200
250
300
100 200 300
50
100
150
200
250
300
100 200 300

SAM

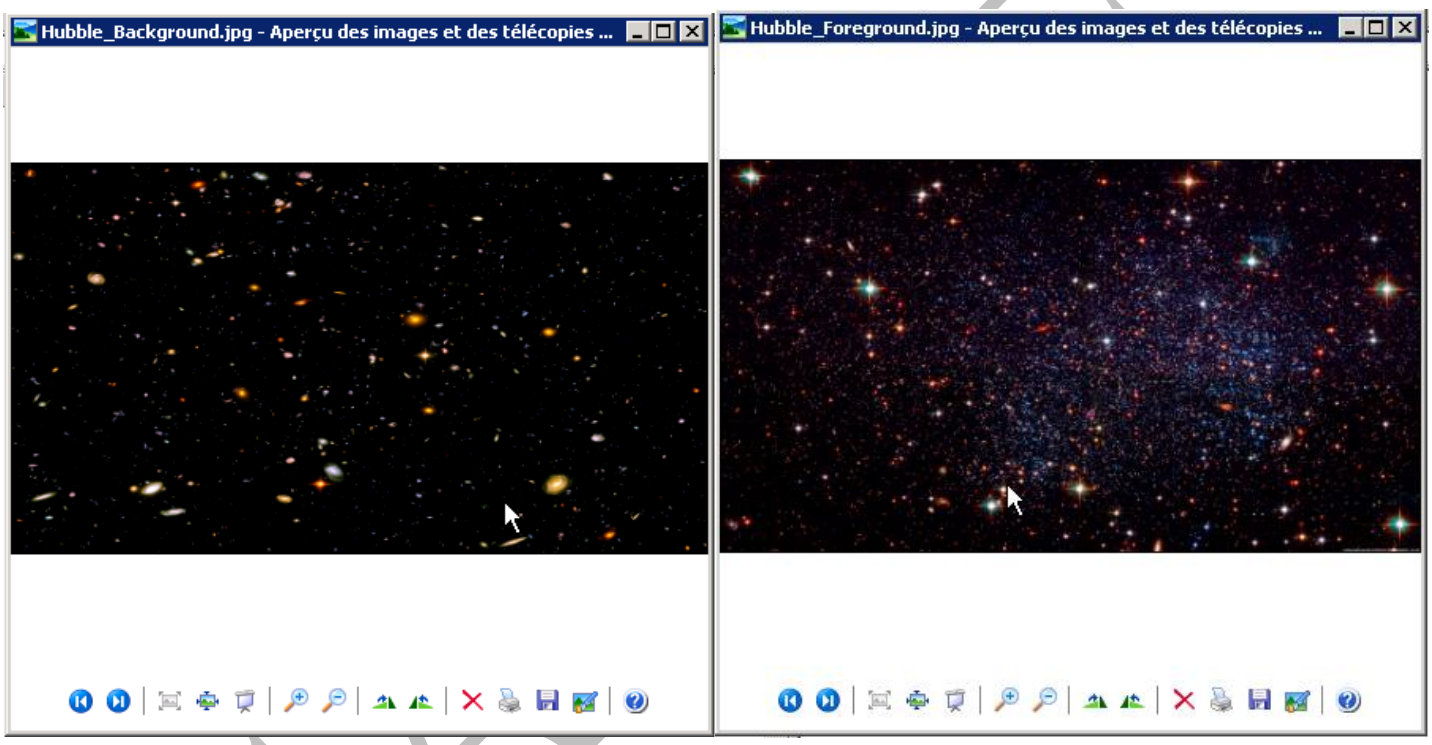
Add (superpose) images

The superposition of images is useful in many scientific fields. One of the most common after medical purposes is astronomy when you have more than one image of a same sky area using different spectrum.

Caution! The images (when you don't have the Image Processing Toolbox) must have the same size!

Caution! The black color is transparent by default for MATLAB!

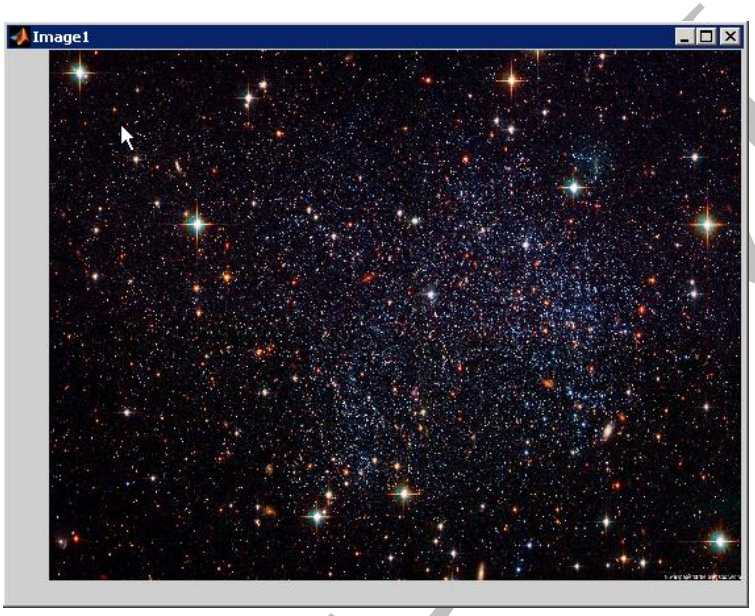
Suppose we have the following images:

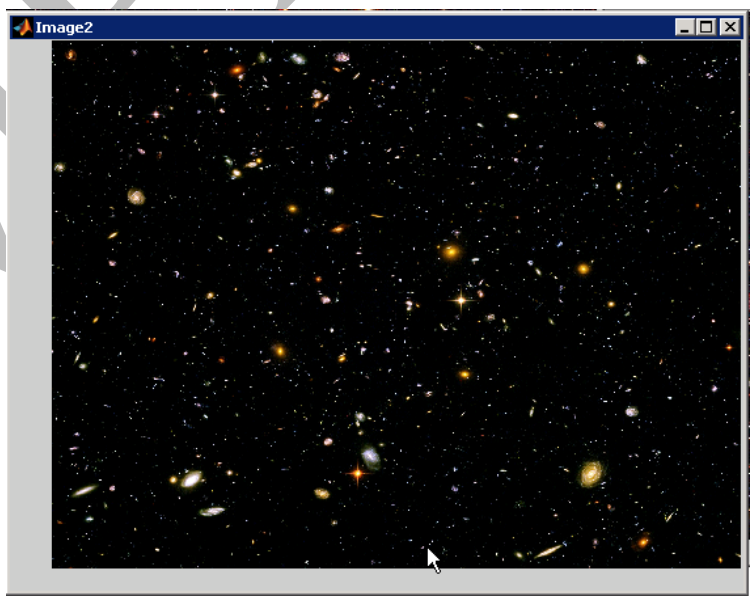


To merge them we just make the sum of the matrices of the both images because black is transparent (0,0,0) by default:

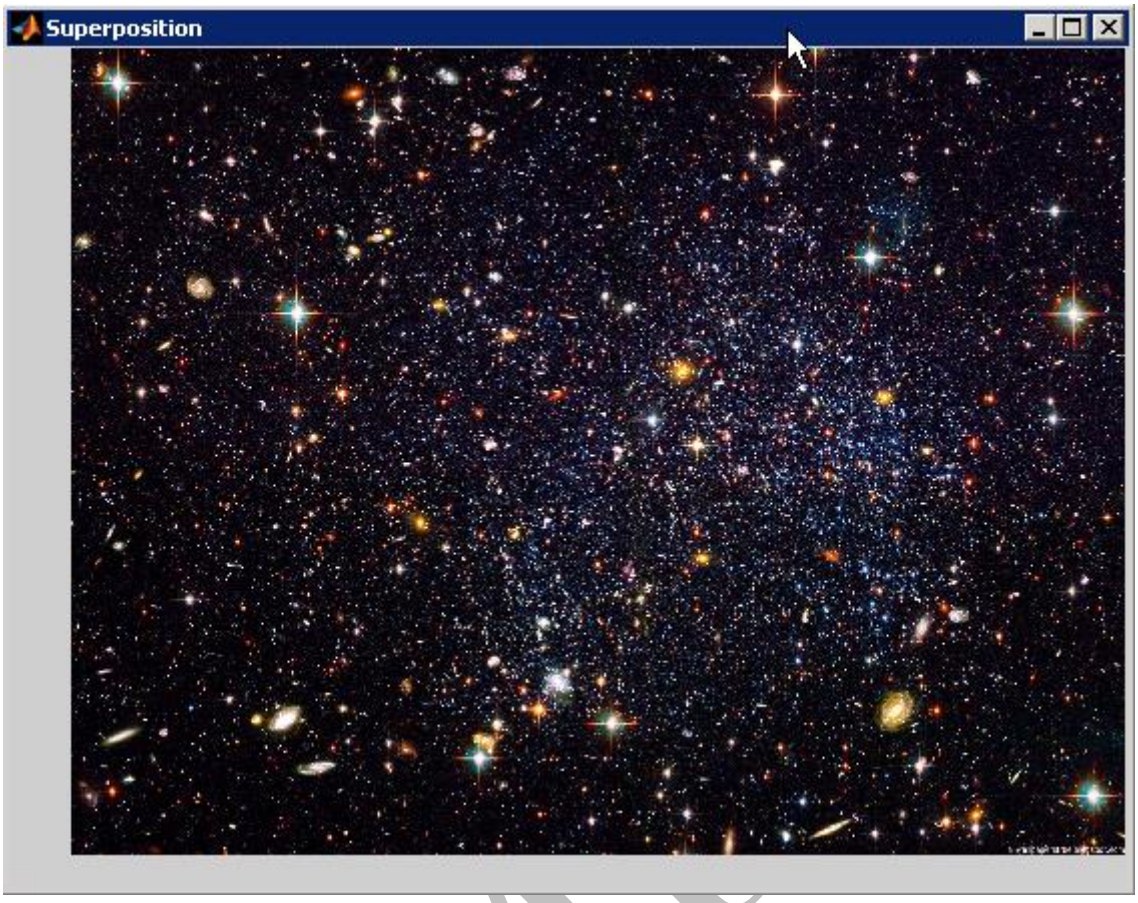
```
1   im1=imread('E:\Hubble_Foreground.jpg');
2   im2=imread('E:\Hubble_Background.jpg');
3   ImageSum=im1+im2;
4   image(im1);
5   set(gcf,'Menubar','none','Name','Image1','NumberTitle','off');
6   set(gca,'LooseInset',get(gca,'TightInset'))
7   axis off;
8   figure;
9   image(im2);
10  set(gcf,'Menubar','none','Name','Image2','NumberTitle','off');
11  set(gca,'LooseInset',get(gca,'TightInset'));
12  axis off;
13  figure;
14  image(ImageSum);
15  set(gcf,'Menubar','none','Name','Superposition','NumberTitle','off');
16  set(gca,'LooseInset',get(gca,'TightInset'));
17  axis off;
```

Then we get first for the two original images:





And for the final result (**ToDo**: find a way to remove MATLAB plot margins...):

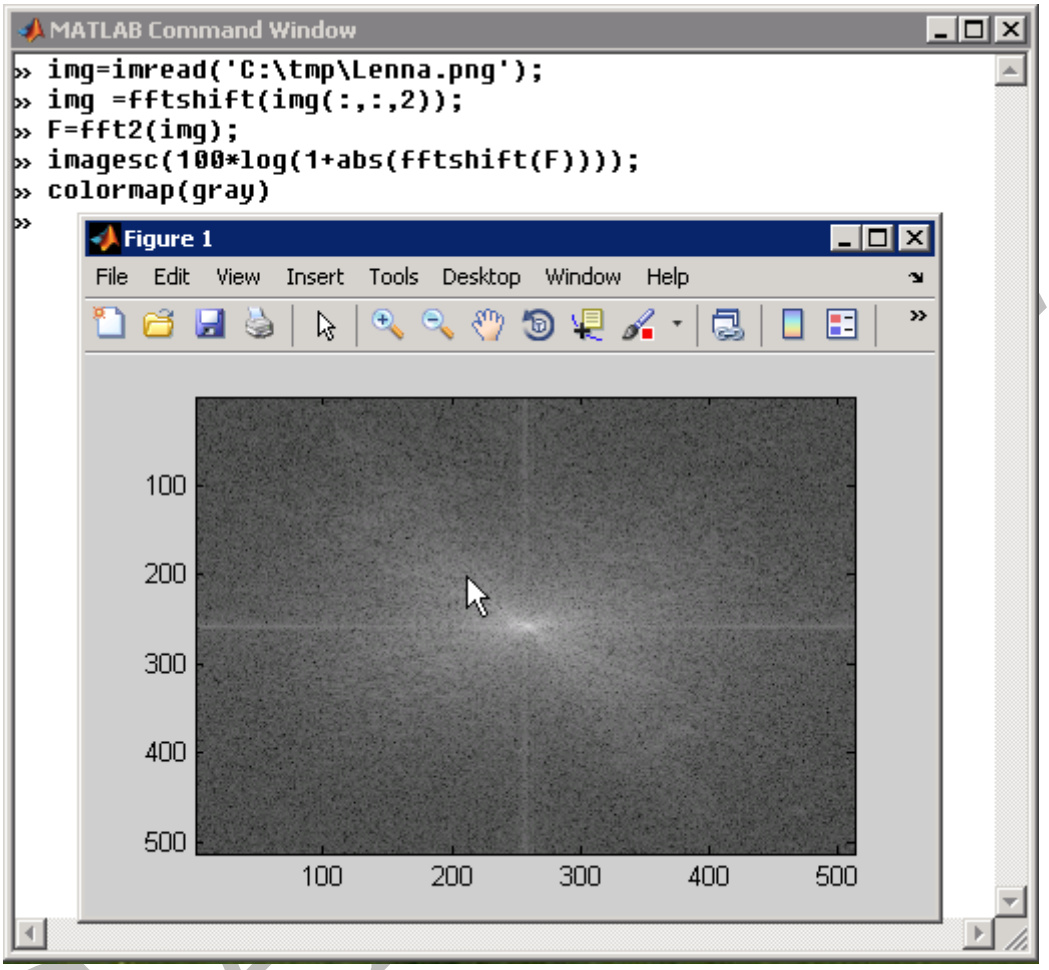


SAMP

2D FFT

We have never calculated by hand in the theoretical course a 2D FFT because it is boring but let us still have a look how to perform an FFT on the Lenna image:

So for this purpose we use the native functions **fftshift( )** and **fft2( )** to get:

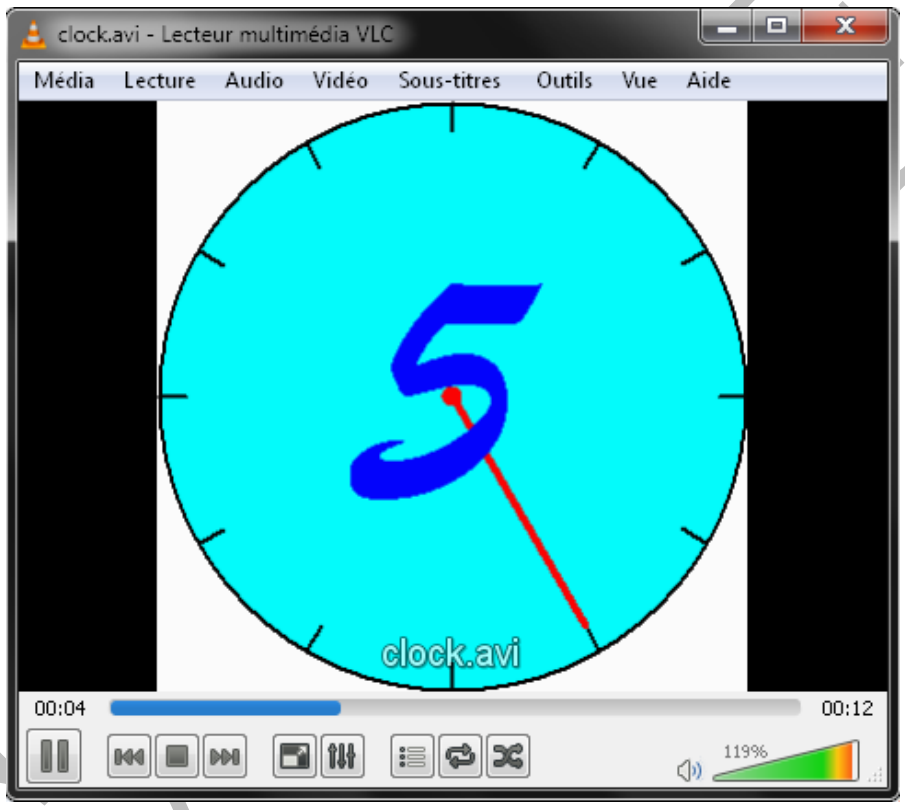


Video processing

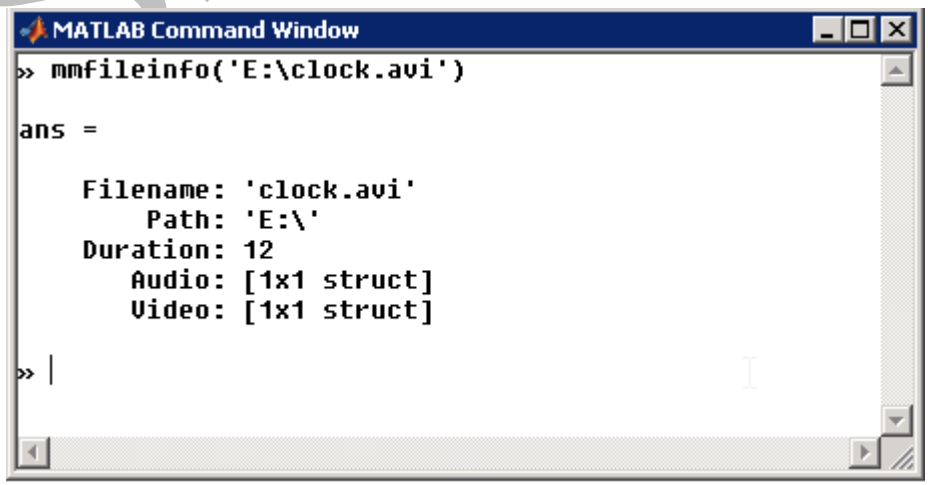
Without the Image Processing Toolbox working with videos in MATLAB is also possible but requires a lot of manipulations of matrices on each video frame to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without to much pain using native core function.

Get video properties

The basics of working with videos are to retrieve information concerning the video file itself. Consider the following video:



We get the properties by using the function **mmfileinfo( )**:

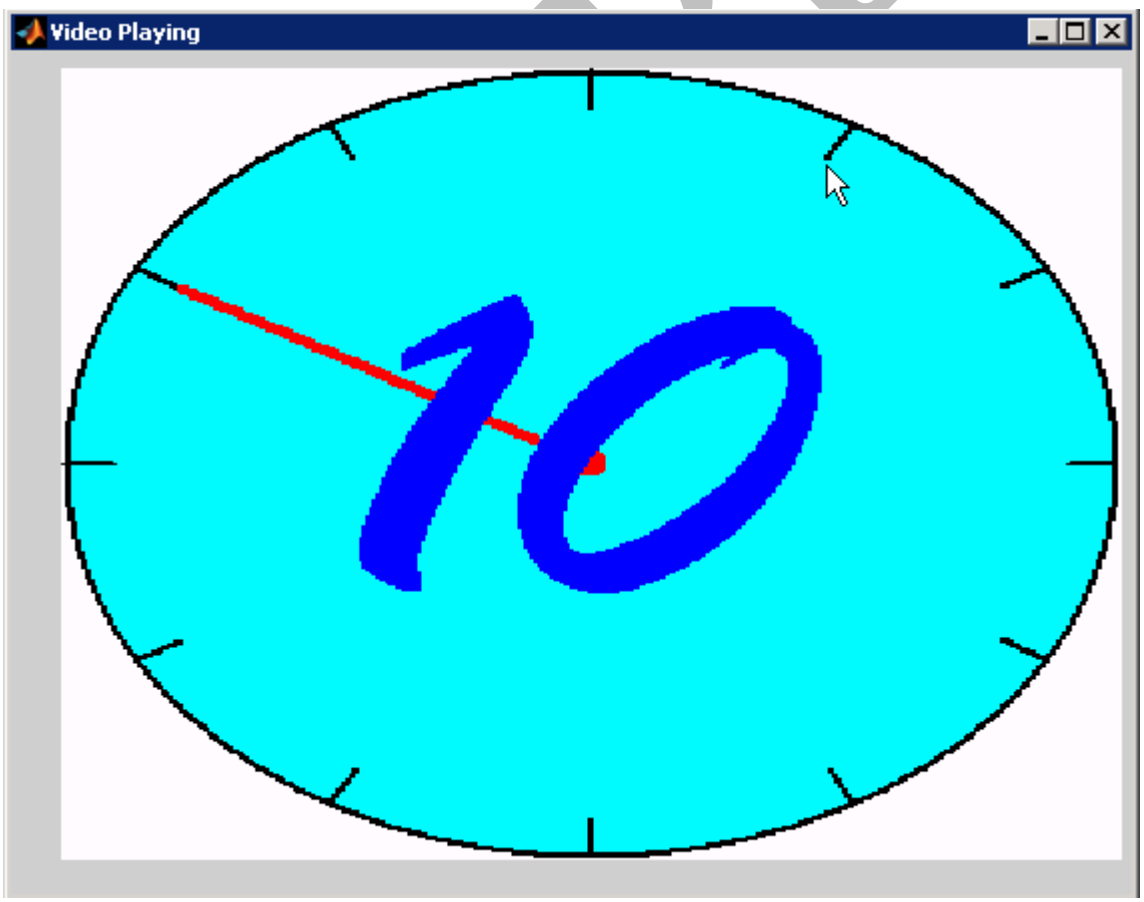


Play video

Without the Image and Visio Processing Toolbox is almost a pain in the a... to play a video. To play we will need to usage the function **VideoReader( )**:

```
1   set(gcf,'Menubar','none','Name','Video Playing','NumberTitle','off');
2   set(gca,'LooseInset',get(gca,'TightInset'))
3   vid=VideoReader('E:\clock.avi');
4   numFrames = vid.NumberOfFrames;
5   n=numFrames;
6   for i=1:n
7       frames = read(vid,i);
8       im(i)=image(frames);
9       axis off;
10  end
```

So the video will play in a MATLAB figure (**ToDo**: found how to remove margins):

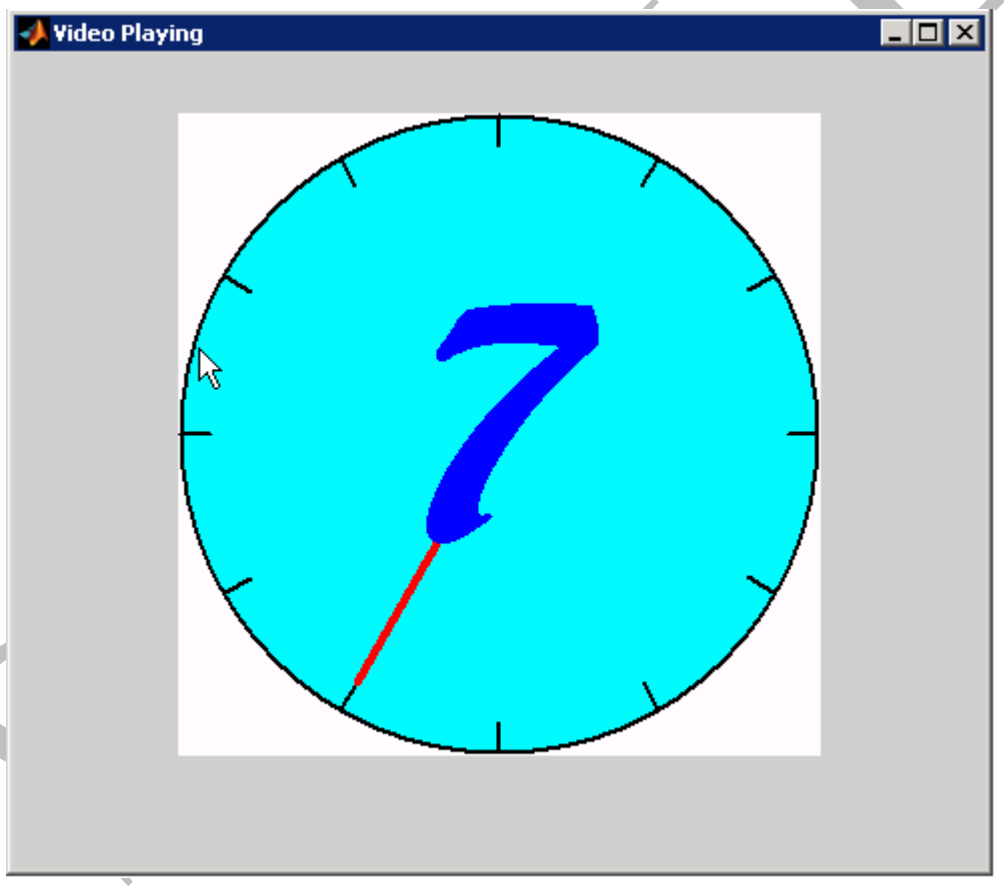


Extract a frame

Extracting a frame from a video is one of the main thing you must learn to do post-processing and this can be done with the conjunctions of the functions **VideoReader( )** and **read( )**:

```
1   vid=VideoReader('E:\clock.avi');
2   im=read(vid,7);
3   imshow(im);
4   set(gcf,'Menubar','none','Name','Video Playing','NumberTitle','off');
5   set(gca,'LooseInset',get(gca,'TightInset'));
```

So we get for the extraction of the 7th frame:



Export all video frames in a folder

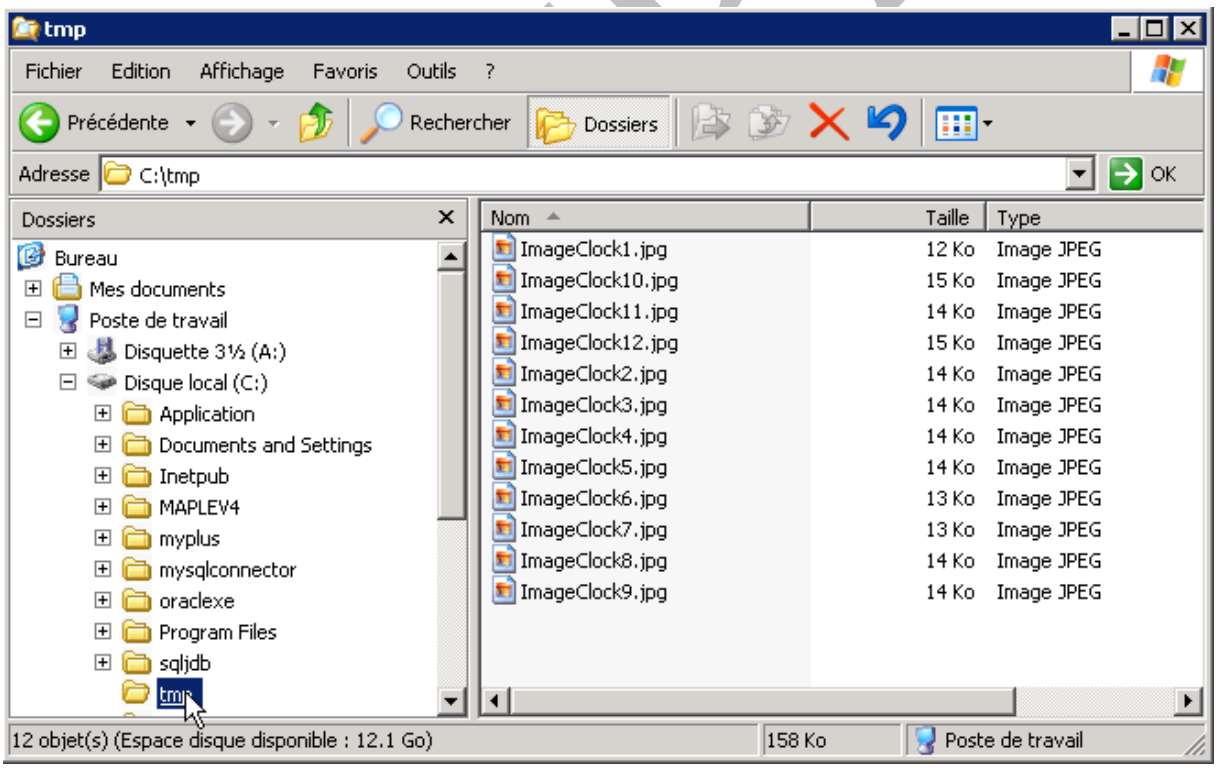
Something also useful sometimes is to decompose a video into multiple frames (images) in a folder for postprocessing purposes.

For this purpose we will use the property **NumberOfFrames** of the method **VideoReader( )** and after the well know **imwrite( )** function:

```
EDITOR  PUBLISH  VIEW

1  vid=VideoReader('E:\clock.avi');
2  numFrames=vid.NumberOfFrames;
3  n=numFrames;
4  for i = 1:n
5      frames = read(vid,i);
6      imwrite(frames,['C:\tmp\ImageClock' int2str(i), '.jpg']);
7      im(i)=image(frames);
8  end
```

This will simply give the expected result:



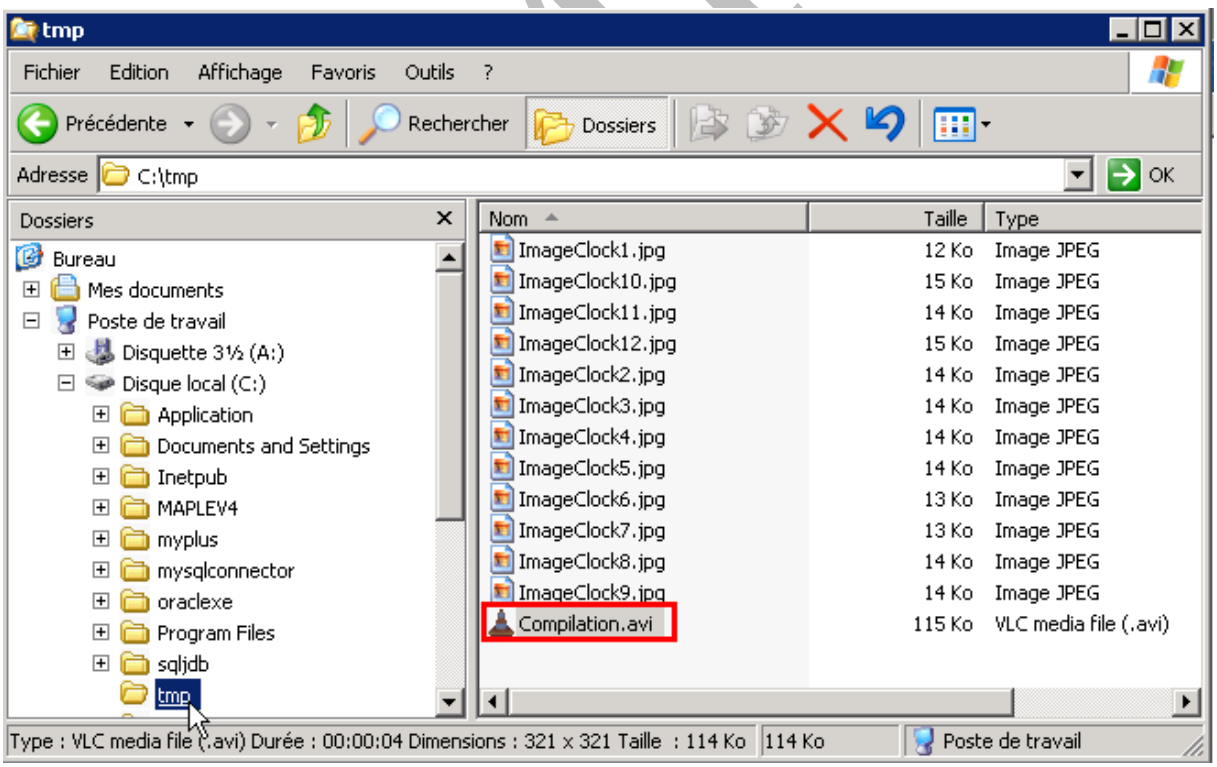
Build a video from a list of (pre-processed or not) images

In the example before we saw how to export frames into a folder for preprocessing. Let us see now how to do the opposite: import images to build a video!

For this purpose we create a new file video using the function **VideoWriter( )** and after we define the images sequence into the file using the function **writeVideo( )**:

```
1   outputVideo=VideoWriter(fullfile('C:/tmp/Compilation.avi'));
2   outputVideo.FrameRate=1;
3   open(outputVideo)
4   ImageList={'C:\ImageClock1.jpg','C:\ImageClock2.jpg',...
5       'C:\ImageClock3.jpg','C:\ImageClock4.jpg'};
6   for k = 1:length(ImageList)
7       img = imread(ImageList{k});
8       writeVideo(outputVideo,img);
9   end
10  close(outputVideo);
```

As you can see below we get a new video file:



And that's it!

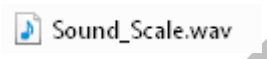
Sound processing

Caution! Nothing will appear on the screen with the example below (no charts, no player control).

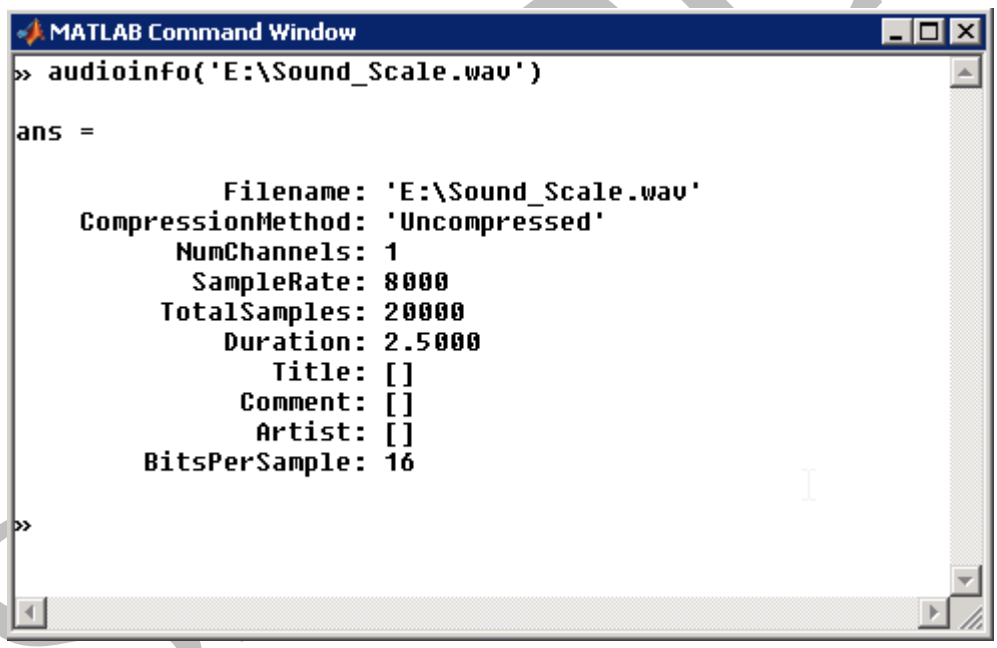
Without the Signal Processing Toolbox working with sounds (signals) in MATLAB is also possible but requires a lot of manipulations of vectoris to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without to much pain using native core function.

Get sound properties

The basics of working with sounds/signals are to retrieve information concerning the signal file itself. Consider the following *.wav file:

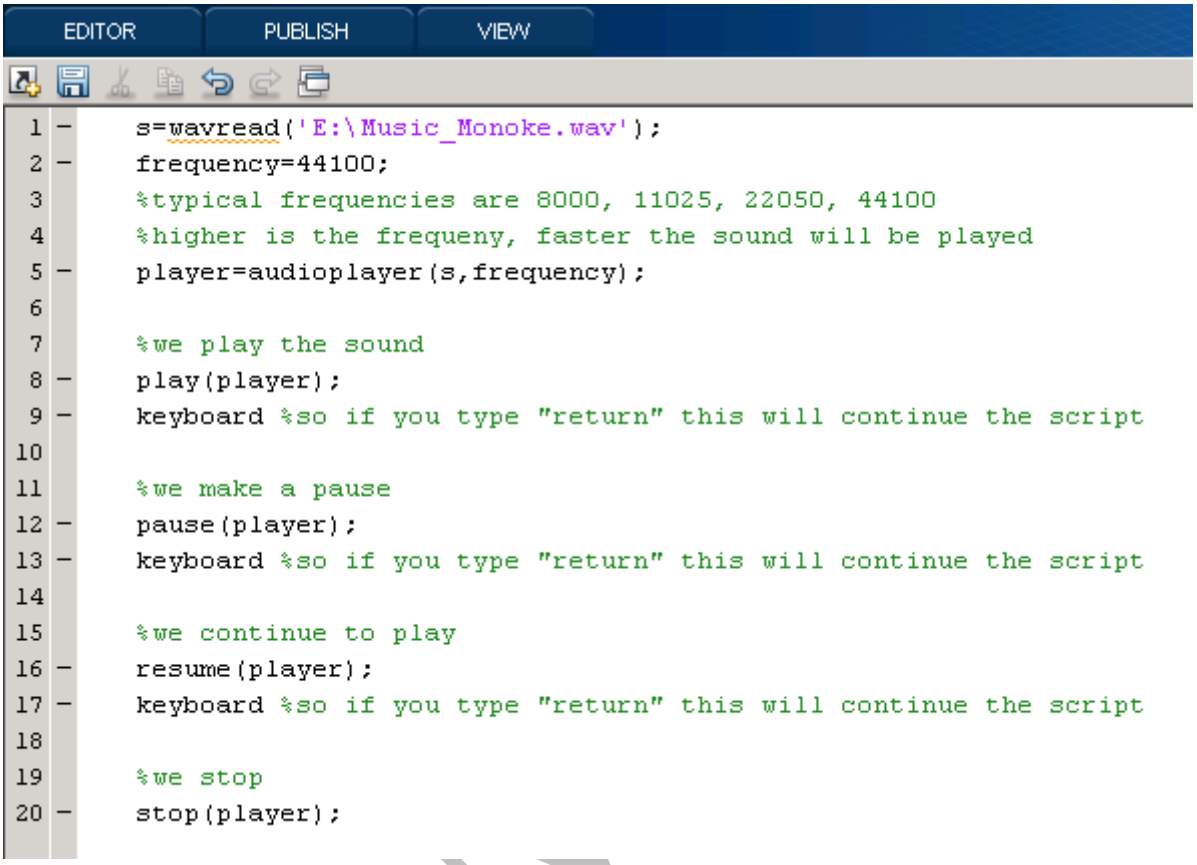


To get information about the content we use the function **audioinfo( )**:



Play, Pause (resume), Resume, Stop sound playing

How using the functions **audioplayer( )**, **play( )**, **pause( )**, **resume( )** and **stop( )** we will see how to play, pause, continue and stop playing a sound/signal:




```
1 -    s=wavread('E:\Music_Monoke.wav');
2 -    frequency=44100;
3      %typical frequencies are 8000, 11025, 22050, 44100
4      %higher is the frequeny, faster the sound will be played
5 -    player=audioplayer(s,frequency);
6
7      %we play the sound
8 -    play(player);
9 -    keyboard %so if you type "return" this will continue the script
10
11     %we make a pause
12 -   pause(player);
13 -   keyboard %so if you type "return" this will continue the script
14
15     %we continue to play
16 -   resume(player);
17 -   keyboard %so if you type "return" this will continue the script
18
19     %we stop
20 -   stop(player);
```

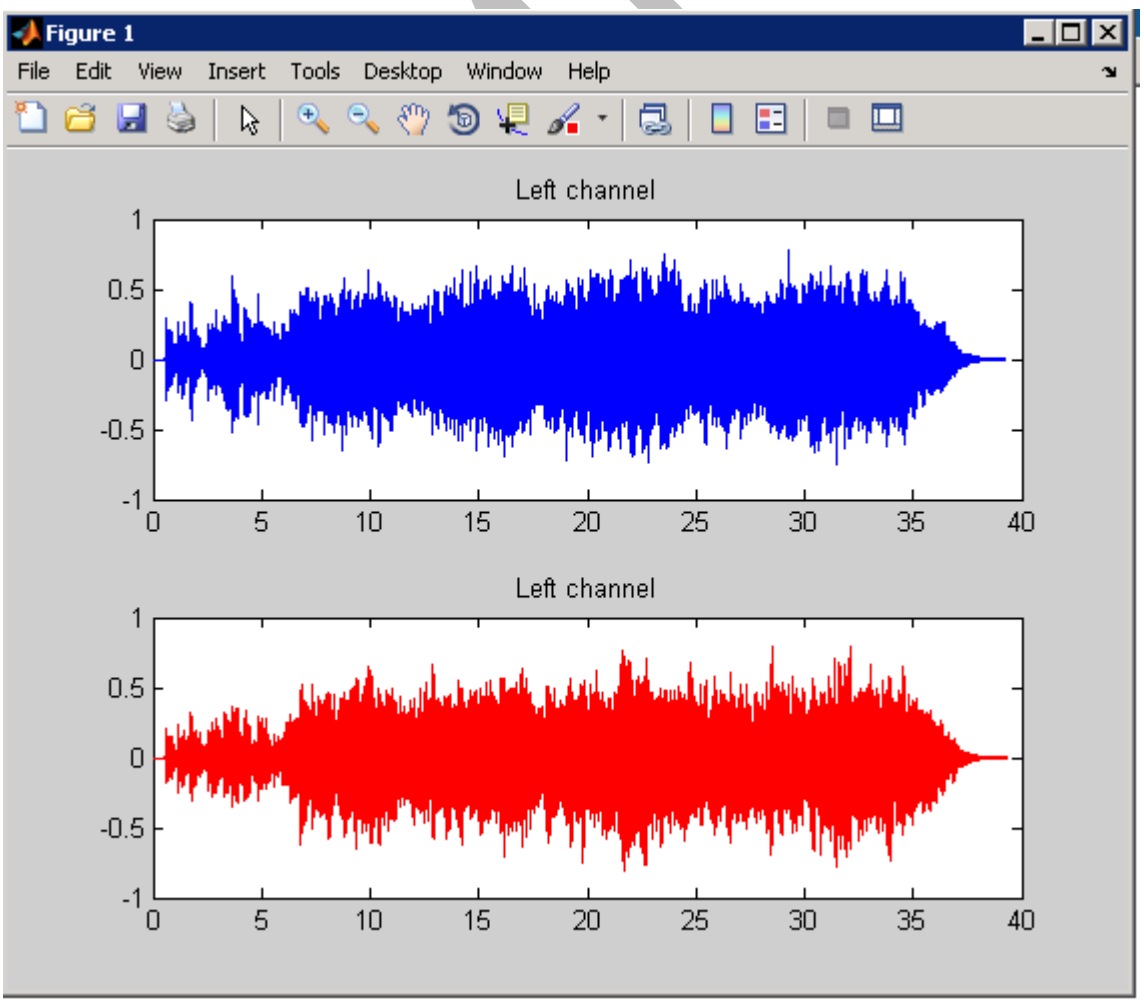
SAM

Plot a sound spectrum

Plot a sound spectrum is one of the most common requests of MATLAB signal processing engineers. Without Toolbox the beginning of the procedure is not necessarily intuitive because it required to use the sampling frequency that we can retrieve using the **wavread( )** function as you can see below

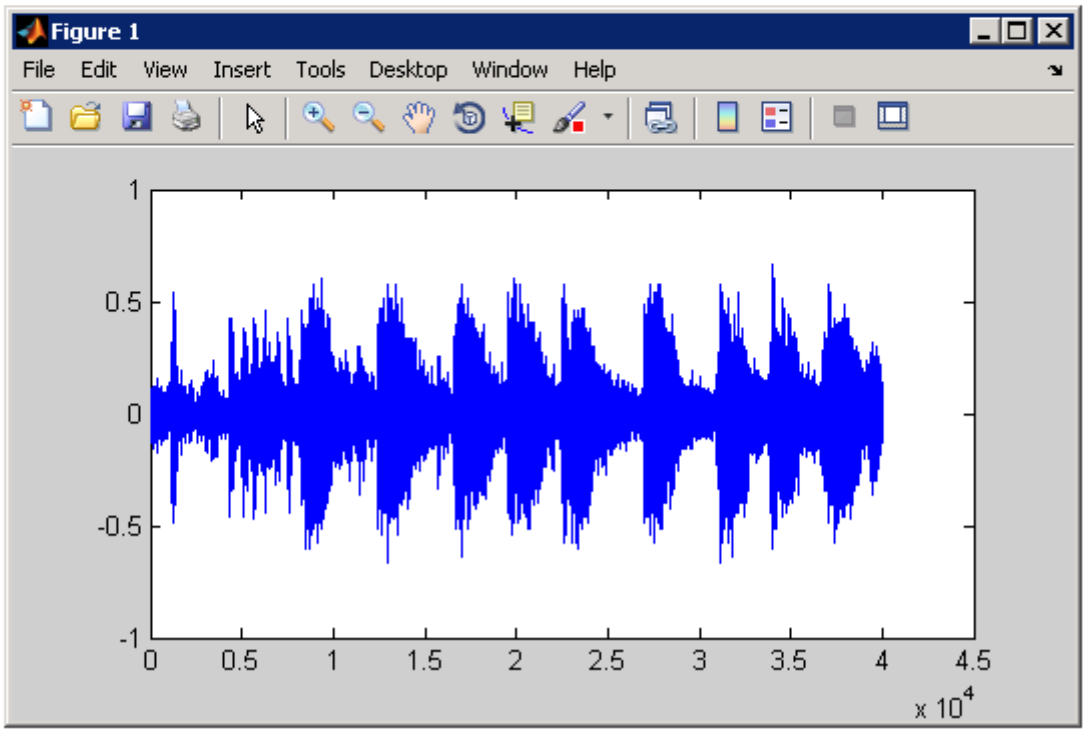
```
1  -    [wave,fs]=wavread('E:\Music_Monoke.wav');
2  -    t=0:1/fs:(length(wave)-1)/fs;
3       %this sound is stereo the the "wave" is a matrix with two column
4       %this is why the plot has two spectrum (one in blue the other in red)
5  -    subplot(2,1,1)
6  -    plot(t,wave(:,1));
7  -    title('Left channel');
8  -    subplot(2,1,2)
9  -    plot(t,wave(:,2),'r');
10 -    title('Left channel');
```

Then we get of a stereo sound signal:



Or another way to do that:

```
1  s=wavread('E:\Music_JamesBond.wav');
2  s_info=audioinfo('E:\Music_JamesBond.wav'); %this sound is mono!
3  length(s);
4  length(s)/s_info.SampleRate; %so we get the 32 seconds!
5  %we extract a part of the sound only
6  s_seg=s(8000*5:8000*10);
7  plot(s_seg);
```

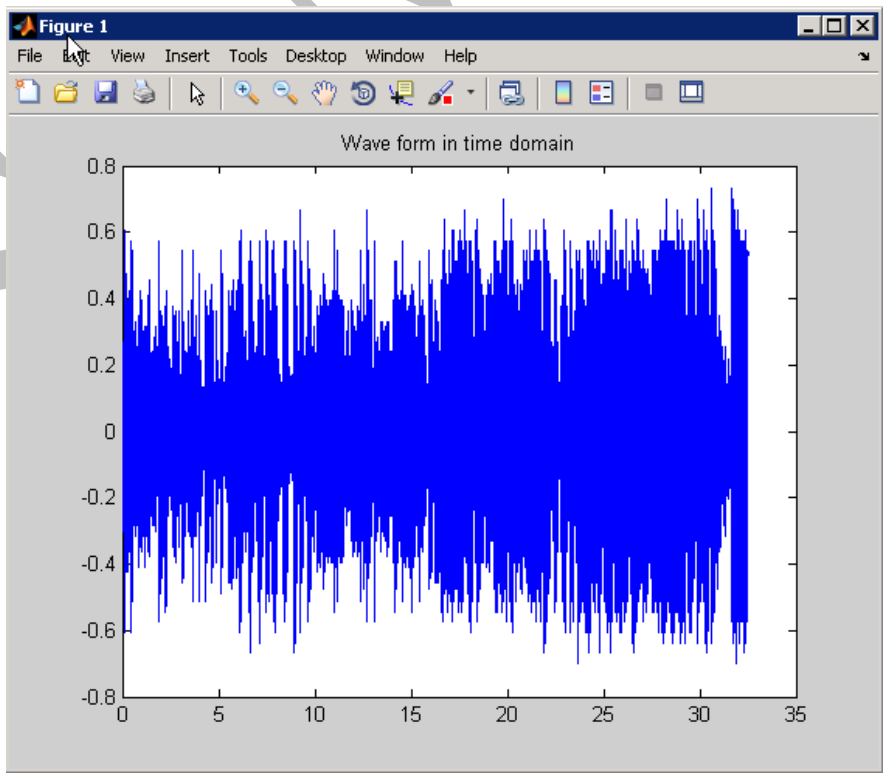


Sound FFT (DFT)

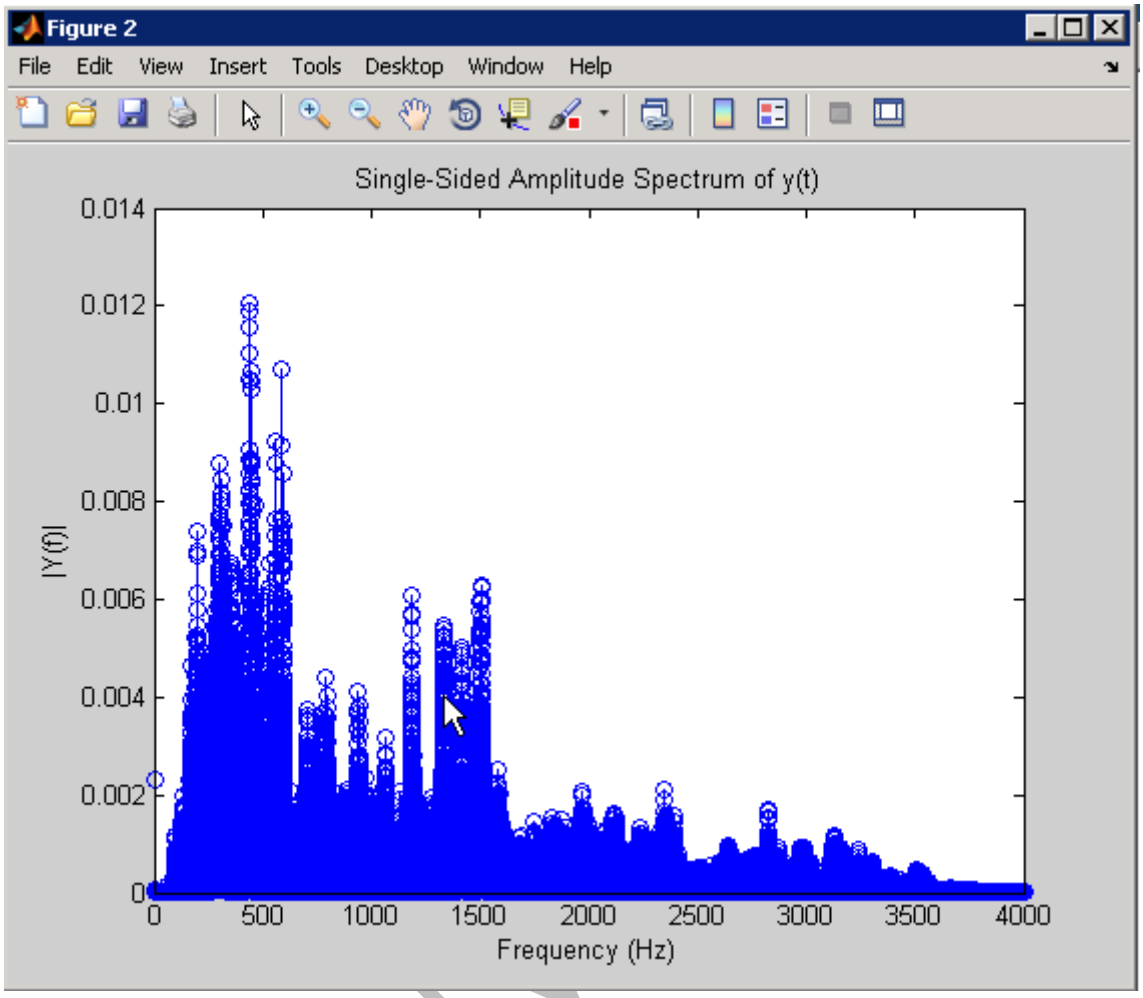
We have already seen how to make an FFT with a random signal (see page 508). The only purpose here is to see a simple way to the same with a sound signal using once again the function **fft( )** and **nextpow2( )**:

```
1   y=wavread('E:\Music_JamesBond.wav');
2   Fs=8000; %Sampling frequency
3   T=1/Fs; %Sample time
4   L=length(y); %Length of signal
5   t=(0:L-1)*T; %Time vector
6   NFFT=2^nextpow2(L); %Next power of 2 from length of y
7   Y=fft(y,NFFT)/L;
8   f=Fs/2*linspace(0,1,NFFT/2+1);
9   figure(1)
10  plot(t,y)
11  title('Wave form in time domain')
12  %Plot single-sided amplitude spectrum.
13  figure(2)
14  stem(f,2*abs(Y(1:NFFT/2+1)))
15  title('Single-Sided Amplitude Spectrum of y(t)')
16  xlabel('Frequency (Hz)')
17  ylabel('|Y(f)|');
```

The original sound signal is:



And the corresponding FFT (DFT) gives:



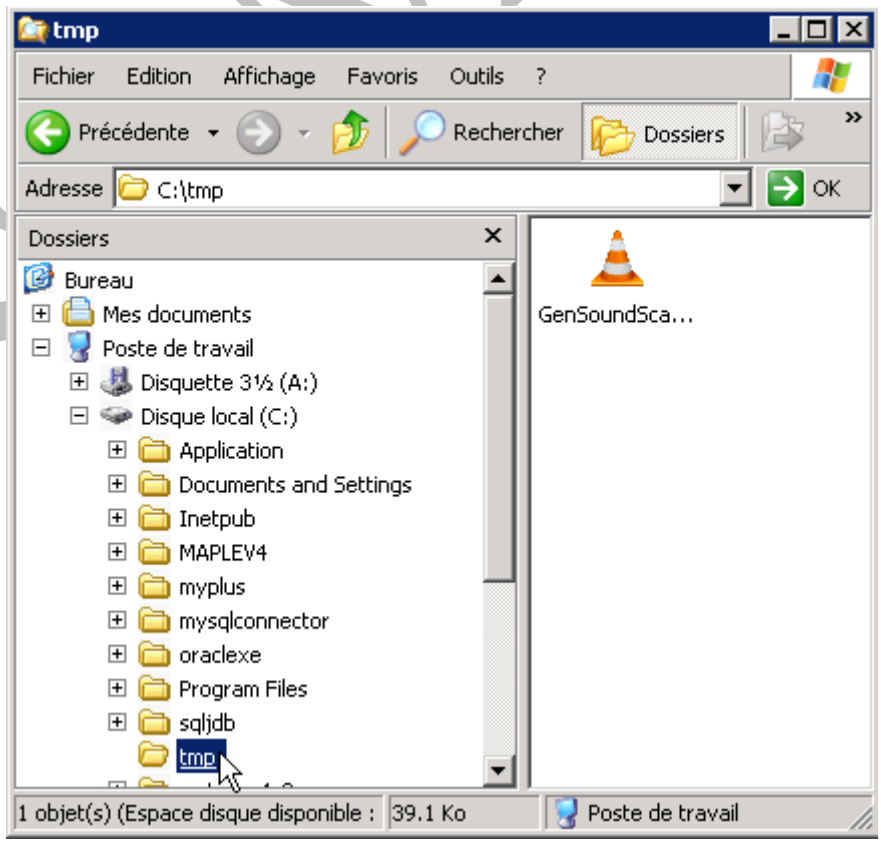
SAM

Generate a sound and save it

Create artificial sounds is a major subject and business in CAM (Computer Assisted Music). We will see here a small academic example on how to play a sound scale using the function **sound( )** and the function **wavwrite( )** to save it on the computer:

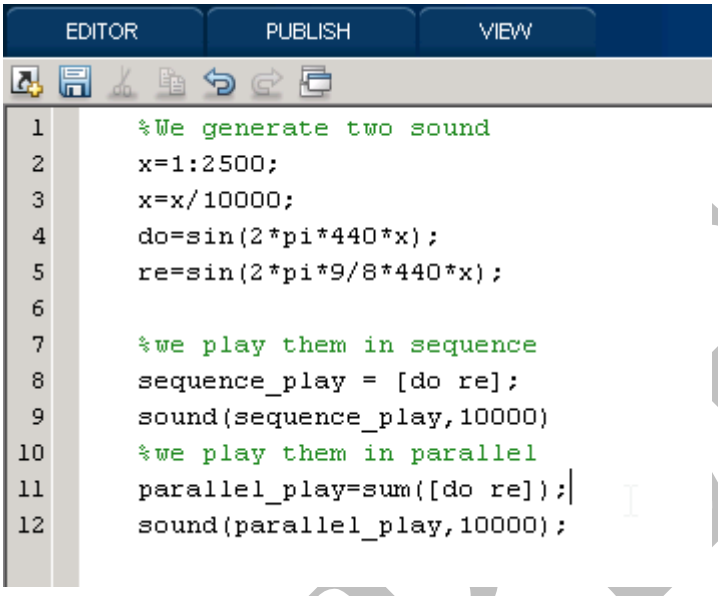
```
%We generate a sound scale
x=1:2500;
x=x/10000;
do=sin(2*pi*440*x);
re=sin(2*pi*9/8*440*x);
mi=sin(2*pi*5/4*440*x);
fa=sin(2*pi*4/3*440*x);
so=sin(2*pi*3/2*440*x);
la=sin(2*pi*5/3*440*x);
si=sin(2*pi*15/8*440*x);
do2=sin(2*pi*2*440*x);
z=[do re mi fa so la si do2];
%we play the sound
sound(z,10000);
%we save the sound
wavwrite(z,'C:\tmp\GenSoundScale.wav')
```

This will create a file as you can see below:



Play sounds in sequence or parallel

In the example before we have play a sound in sequence (serial) but what about playing two or more sounds in parallel using the function **sound( )**?:



SAMPLE

Scripting

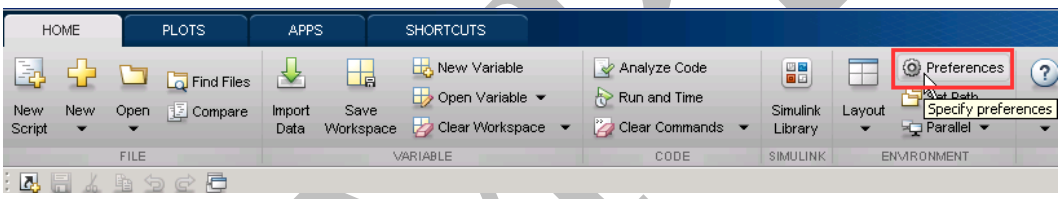
Caution! New releases of MATLAB have autocompletion/intellisense for built-in functions but not for contextual variables/functions that you created yourself!

MATLAB script is the major subject of MATLAB as for any scientific tool. Scripts gives you the possibility to automate analysis, trading orders, machine behaviors, etc. The most important fact in MATLAB is that you can at the end compile you script in a C++ ou C# *.dll and use the resulting code in your own applications.

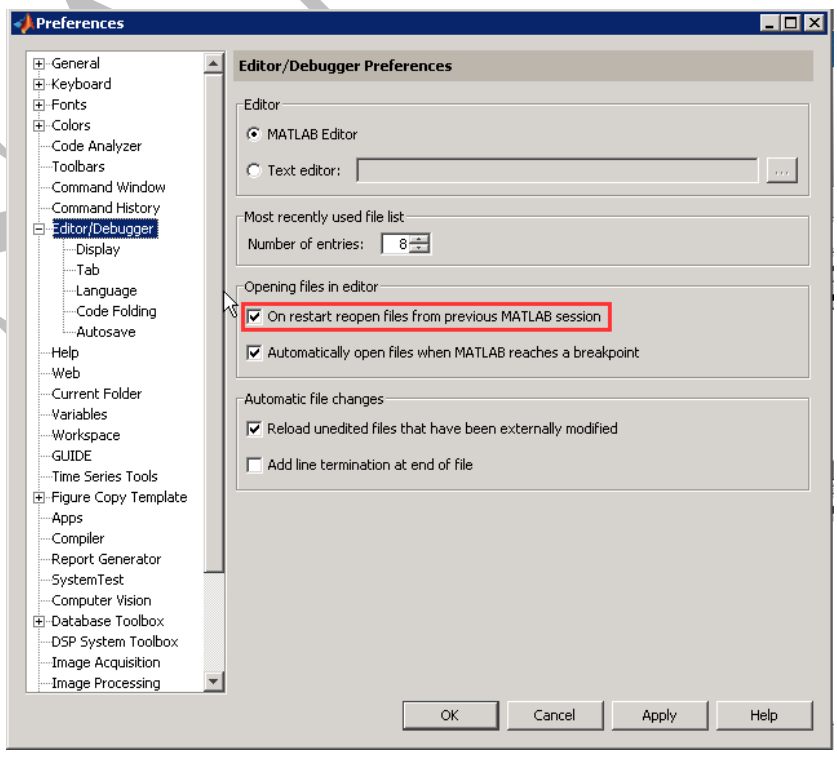
The MATLAB script language is not a programming language, you do not have a difference between routine and functions and the variables are not signed but you must always keep in mind the purpose of the R&D engineer is not to become and computer scientist but to have a RAD (rapid application development) solution to not lose too much time on computer and focus more on lab stuff.

Script Editor most asked preferences

Let us first begin by the most asked preferences questions of MATLAB users about the script editor. In the **Home** tab click on **Preferences**:

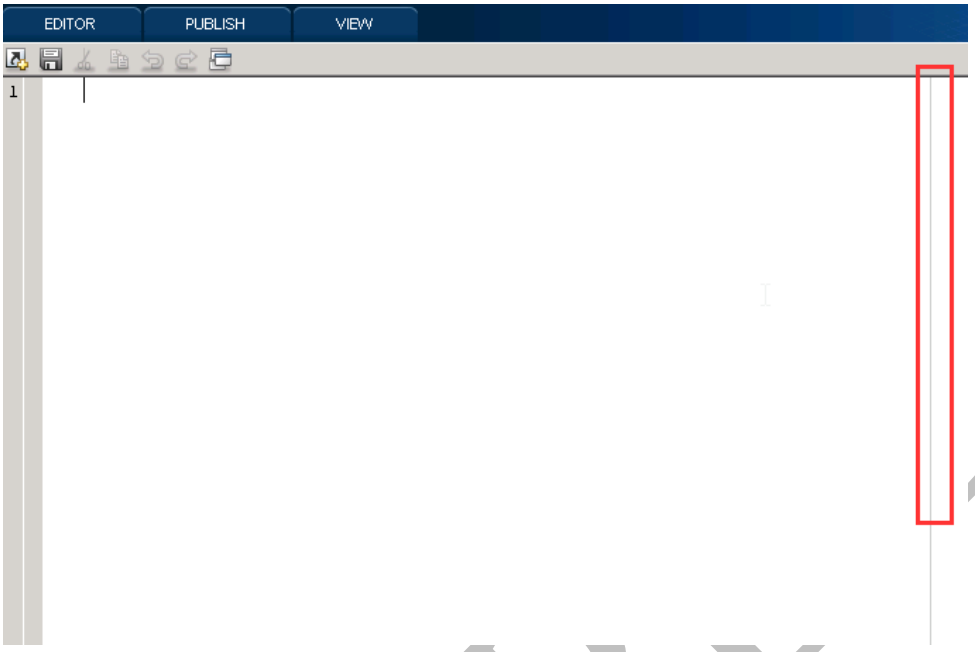


In the Editor/Debugger section you can choose or not to have last script files to be reopen at each new MATLAB session:

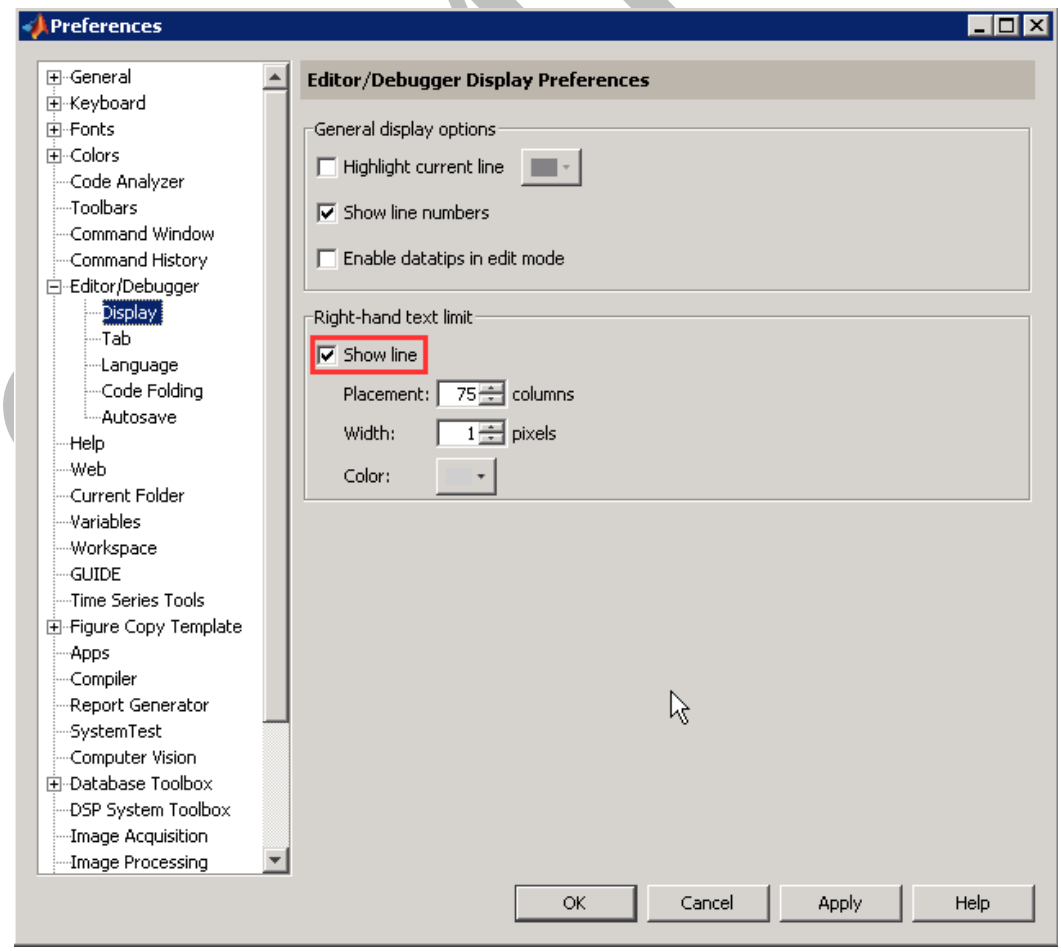


SAMPLE

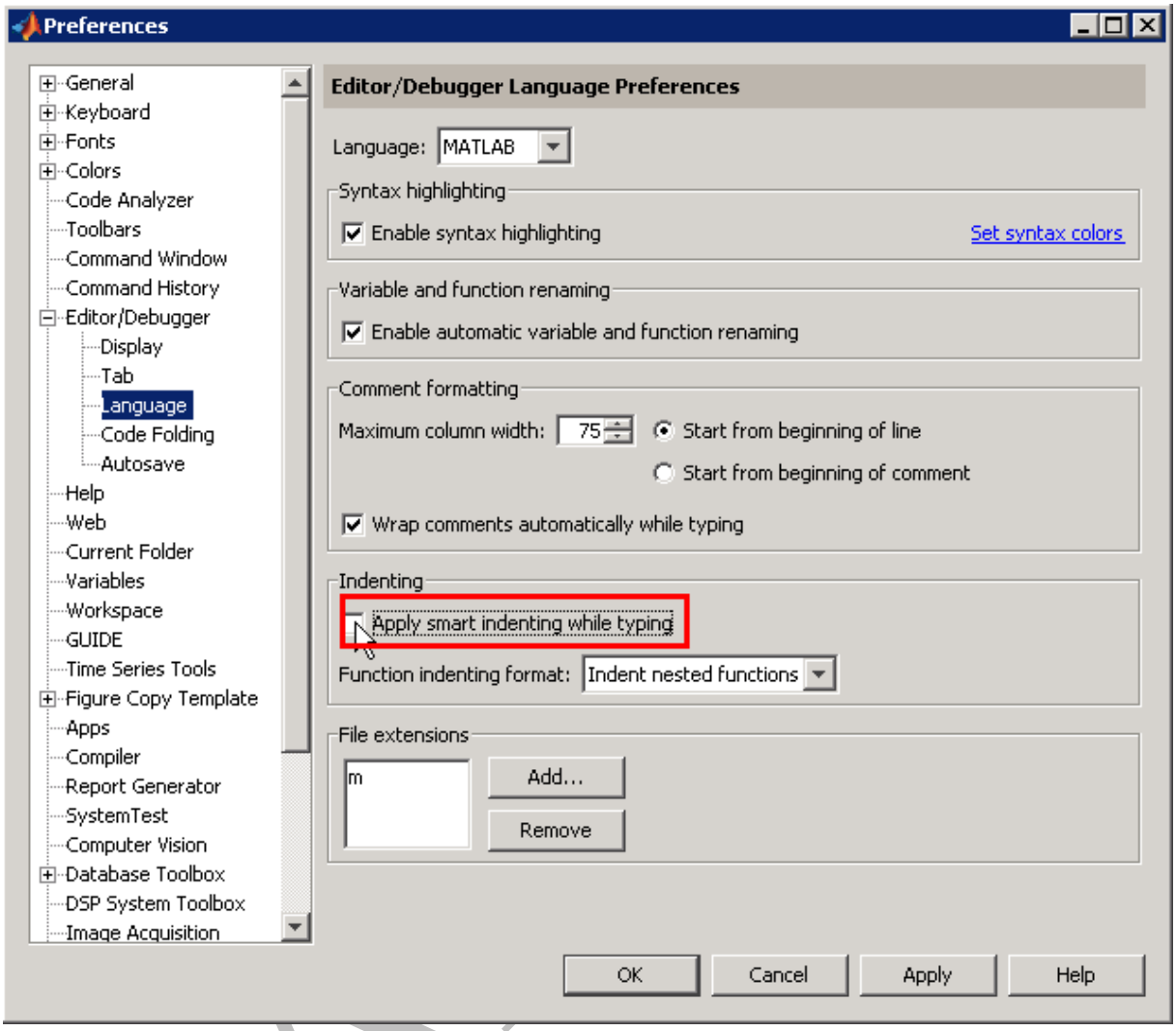
Now something boring in the script editor is the page break line on the right for people that print scripts on paper or on PDF:



but most user don't print anymore... Then let us see how to remove this trough the section **Editor/Debugger** in the subsection **Display** you can deactivate **Show line**:

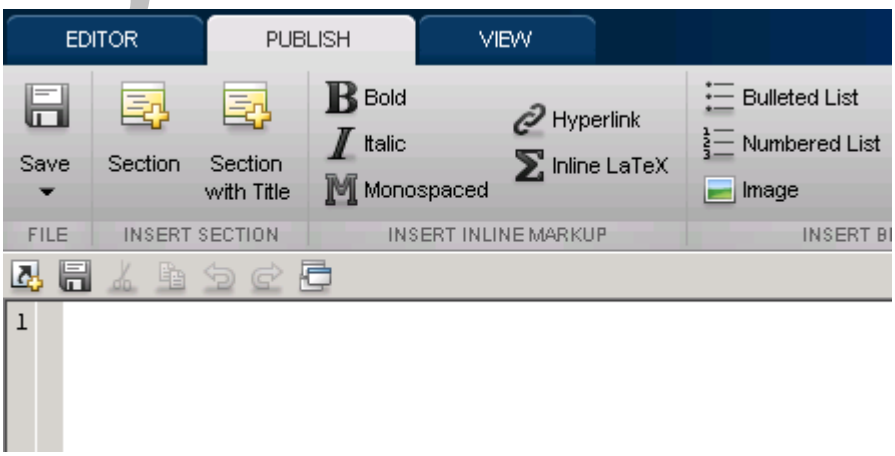


And if you don't want auto-indenting (auto-tabulation) while you are typing MATLAB script just change the following:

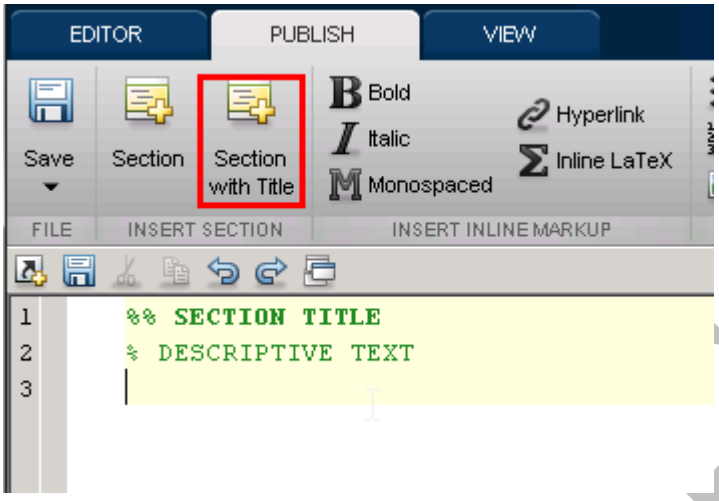


Well commented script

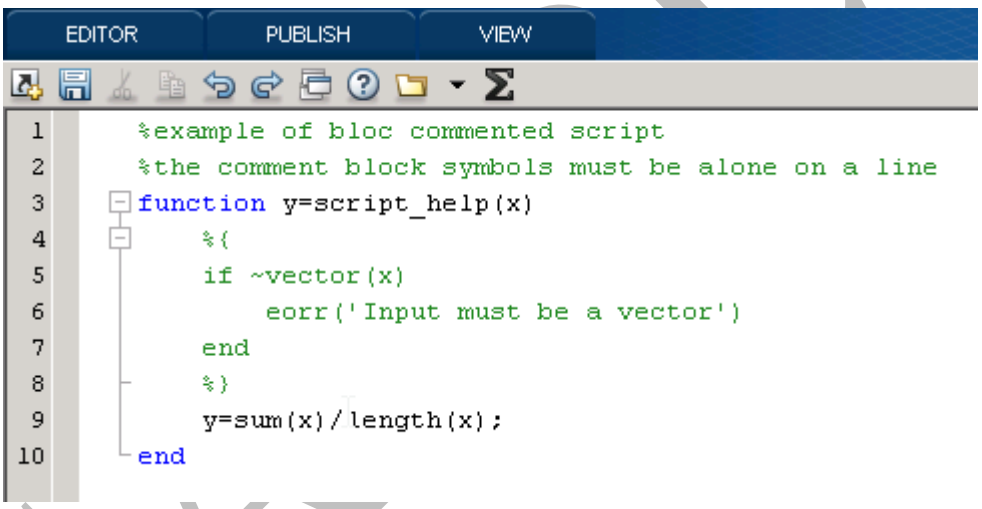
Comment script is a very important process in the field of engineering. Everything that is relative to comments is in the register **Publish**:



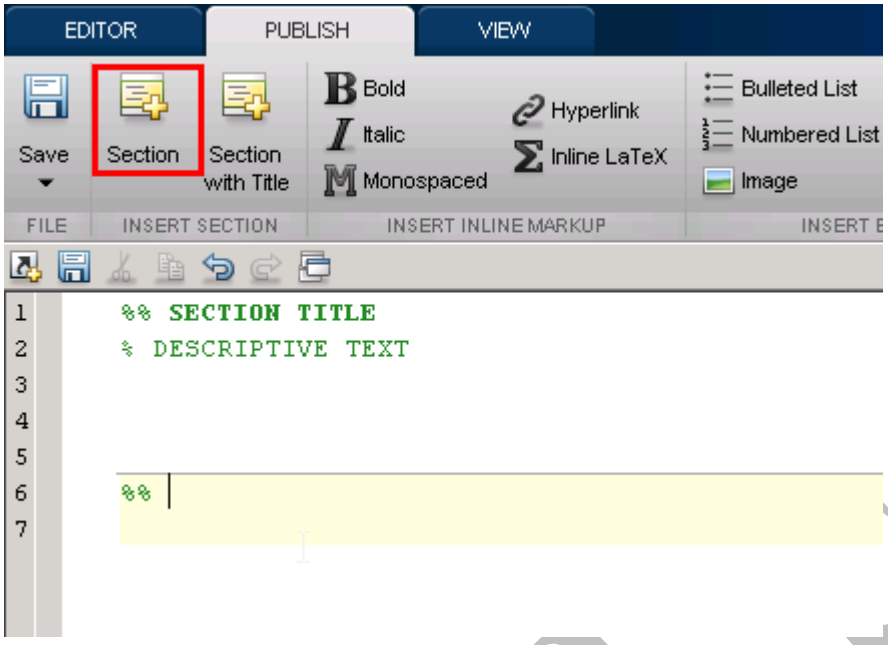
The first thing to do when you create a new script is to insert a **Section with Title**:



Thus you see that to comment a line we just simply write **%** at the beginning of the line or we use the keyboard shortcut **Ctrl+R**. You can also comment a block of script using **%{ ... %}**:



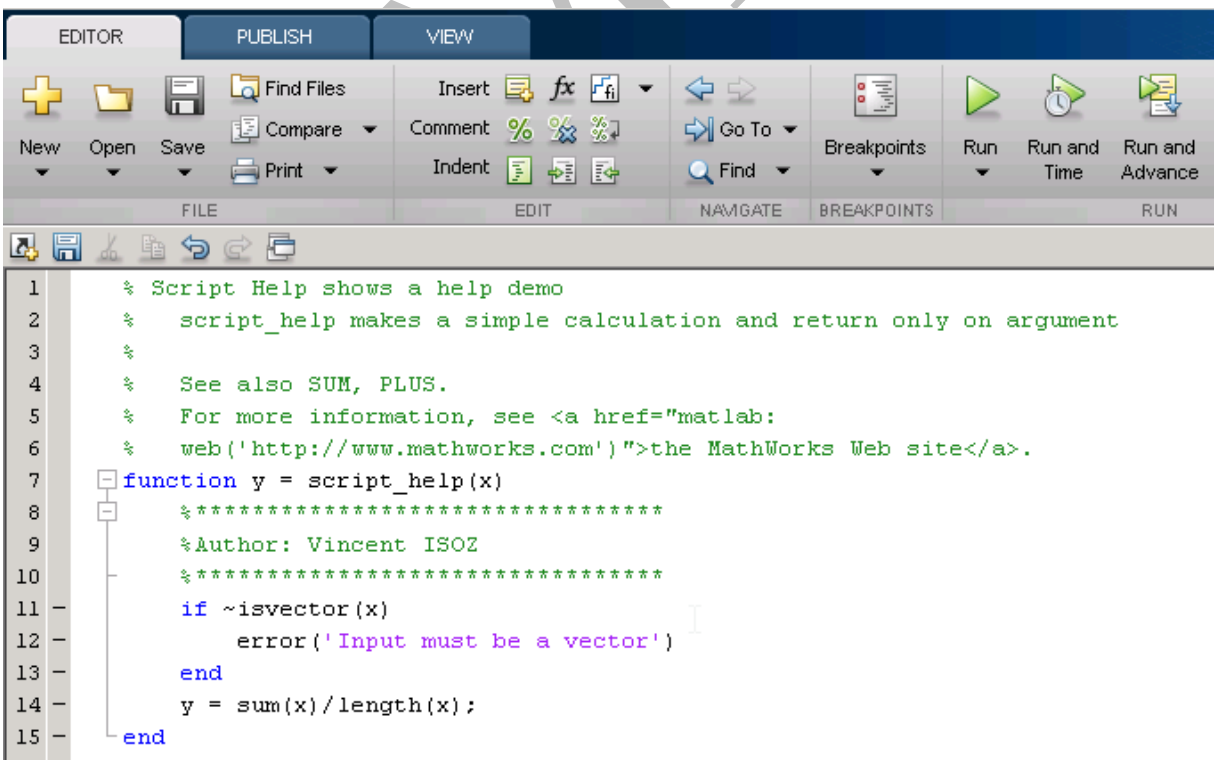
After you can insert anywhere a **Section** (also called "cell") in the code:



Note: If you press **Ctrl+Enter** to execute a code only the active section (active cell) will be run.

Define help text

In MATLAB by default the first comments at the top of the scripts will be used by the **help** command:



As you can see here is demo using simple texts and hyperlinks:

```
Command Window
>> help script_help
  Script Help shows a help demo
    script_help makes a simple calculation and return only on argument

    See also sum, plus.
    for more information, see the MathWorks Web site.

fx >>
```

As you can see nothing is especially difficult. But in facts what we did is false!!

Yes the good method is the following (the script is not the same as above but this is not the subject, just focus on the header part!):

EDITOR PUBLISH VIEW

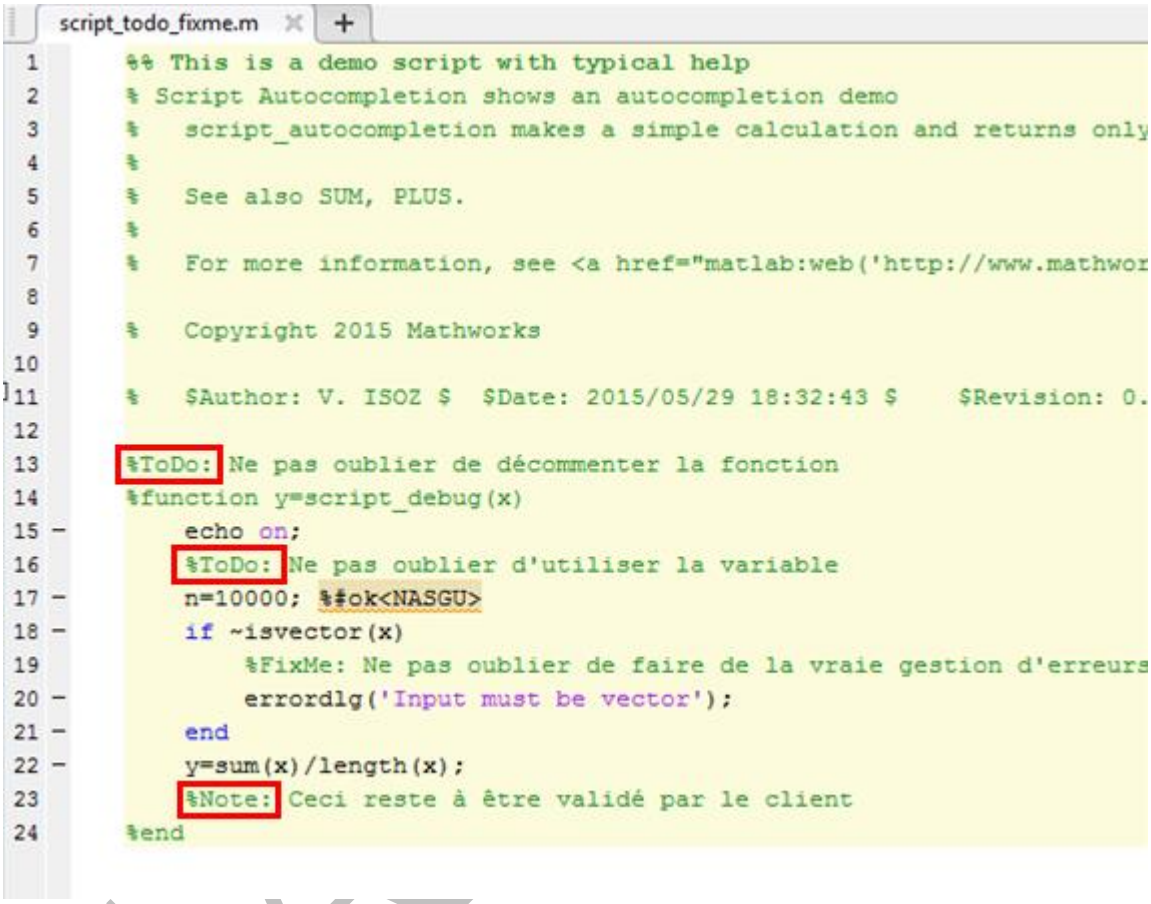
```
1   %%Main title of our demo script to study help syntax
2   %   script_help is a demo script to see how to correctly use MATLAB
3   %   comments
4   %
5   %   Example:
6   %
7   %   ScriptDemo(x) Then you get the following result
8   %
9   %   For more information, see <a href="www.mathworks.com">Alphorm</a>
10  %   See also: SUM, PLUS
11
12  %   Copyright 2014 Mathworks
13
14  %   $Author: V. ISOZ $     $Date: 2014/08/07 18:23:52 $    $Revision: 0.1 $
15
16  function y=script_help(x)
17      x=5; %this is a constant for...
18      %{
19      if ~isvector(x)
20          error('Input must be a vector');
21      end
22      %}
23      Y=sum(x)/length(x);
24  end
```

Since the last versions of MATLAB it seems that the Copyright and Author metadata doesn't work in the publishing tool anymore ☹

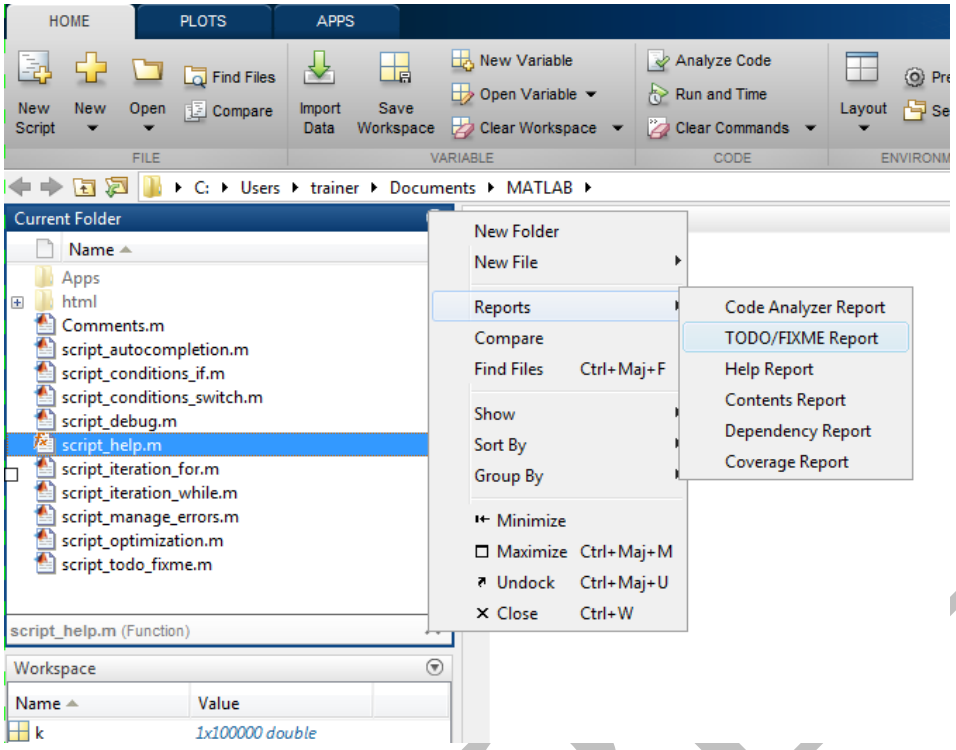
Use ToDo, FixMe and Notes

A major requirement of many developers is to have like Visual Studio .Net the possibility to write at given places of the code comments using the keywords *ToDo*, *FixMe*, *Note* and after to get a summary like a report of the such comment.

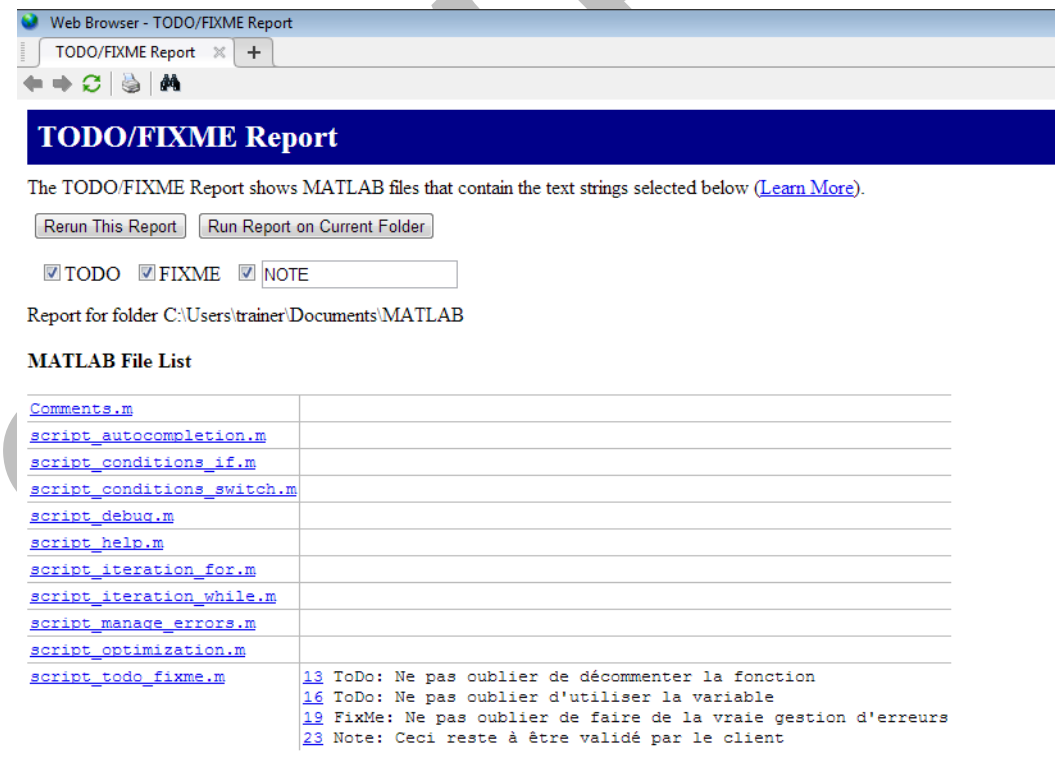
MATLAB is able also to do this. Consider the following example:



Now to get a summary of all such comment, you need to the main MATLAB GUI to open the **TODO/FIXME report**:

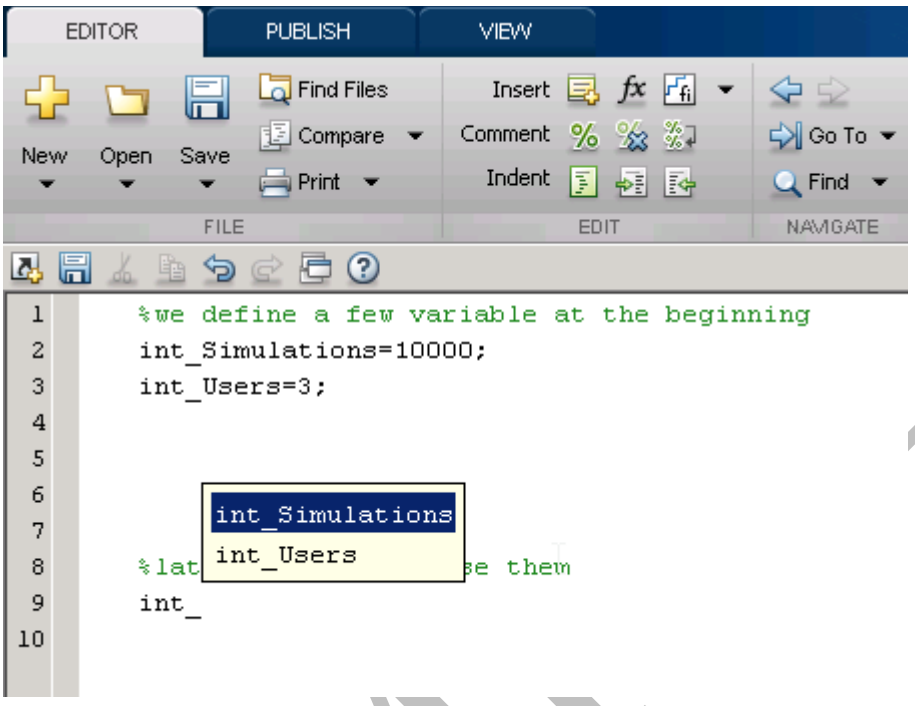

HOME
PLOTS
APPS
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Layout
FILE
VARIABLE
CODE
C: ▸ Users ▸ trainer ▸ Documents ▸ MATLAB ▸
Current Folder
Name
Apps
html
Comments.m
script_autocompletion.m
script_conditions_if.m
script_conditions_switch.m
script_debug.m
script_help.m
script_iteration_for.m
script_iteration_while.m
script_manage_errors.m
script_optimization.m
script_todo_fixme.m
script_help.m (Function)
New Folder
New File
Reports
Compare
Find Files Ctrl+Maj+F
Show
Sort By
Group By
Minimize
Maximize Ctrl+Maj+M
Undock Ctrl+Maj+U
Close Ctrl+W
Code Analyzer Report
TODO/FIXME Report
Help Report
Contents Report
Dependency Report
Coverage Report
Workspace
Name
Value
k
1x100000 double

So you get:

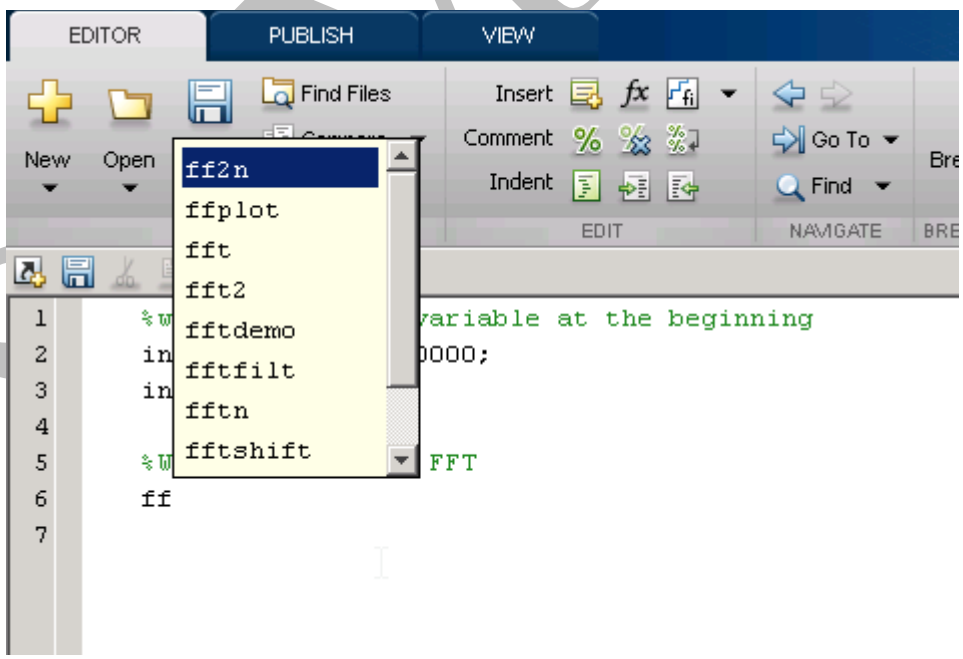

Web Browser - TODO/FIXME Report
TODO/FIXME Report
TODO/FIXME Report
The TODO/FIXME Report shows MATLAB files that contain the text strings selected below (Learn More).
Rerun This Report
Run Report on Current Folder
TODO
FIXME
NOTE
Report for folder C:\Users\trainer\Documents\MATLAB
MATLAB File List
Comments.m
script_autocompletion.m
script_conditions_if.m
script_conditions_switch.m
script_debug.m
script_help.m
script_iteration_for.m
script_iteration_while.m
script_manage_errors.m
script_optimization.m
script_todo_fixme.m
13 ToDo: Ne pas oublier de décommenter la fonction
16 ToDo: Ne pas oublier d'utiliser la variable
19 FixMe: Ne pas oublier de faire de la vraie gestion d'erreurs
23 Note: Ceci reste à être validé par le client

Autocompletion

If you want to use autocompletion in MATLAB begin to write the variable name/function you want and press the **TAB** button on your keyboard:



Also the same thing works for native functions:



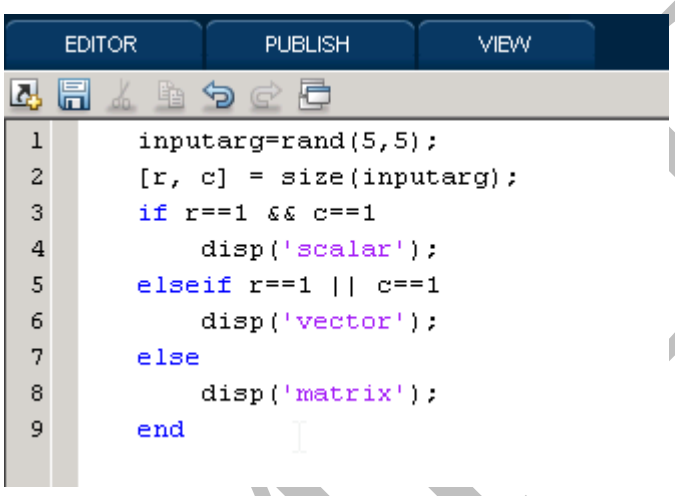
Move with the arrows of your keyboard and press **ENTER** to validate (or **TAB** again).

Conditional structure (if/switch)

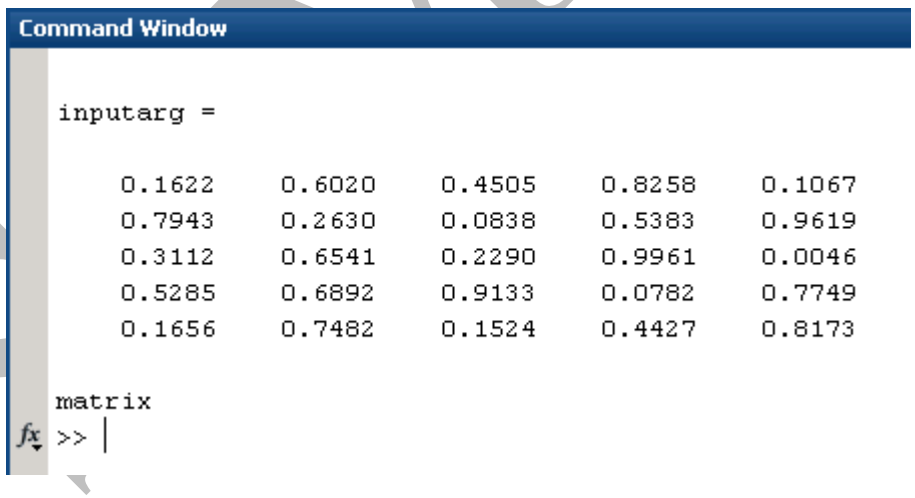
Note: The purpose of this document is not to make an introduction to algorithmic. If you don't know how to make programming/scripts first follow an algorithmic course or read a book on the subject.

if statement

Let us see the most simple and complete example of an **If... Elseif...Else...End** MATLAB statement syntax with OR (||) and AND (&&) operators=



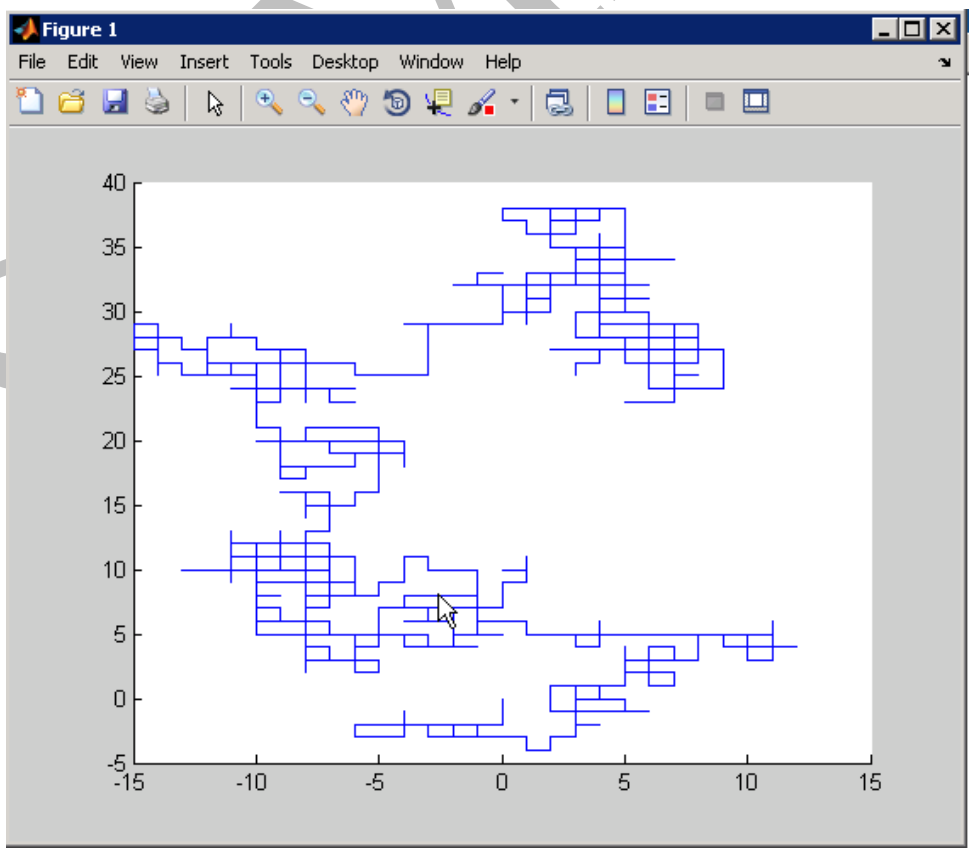
As you can see if we run the script (**Ctrl+Enter**), we get:



And well another well know good exercise for applying if statement is to plot a random walk (Brownian motion):

```
1   r=[0 0];
2   for t= 0:0.1:100
3       rand_step=rand(1,2);
4       B=rand_step>0.5;
5       if B==1
6           rnew=r+[1 0];
7       elseif B(1)==1
8               rnew=r+[0 1];
9       elseif B(2)==1
10          rnew=r+[-1 0];
11      else
12          rnew=r+[0 -1];
13      end
14
15      hold on;
16      plot([r(1) rnew(1)],[r(2) rnew(2)]);
17      r=rnew;
18  end
```

This will result in:



switch statement (with if…)

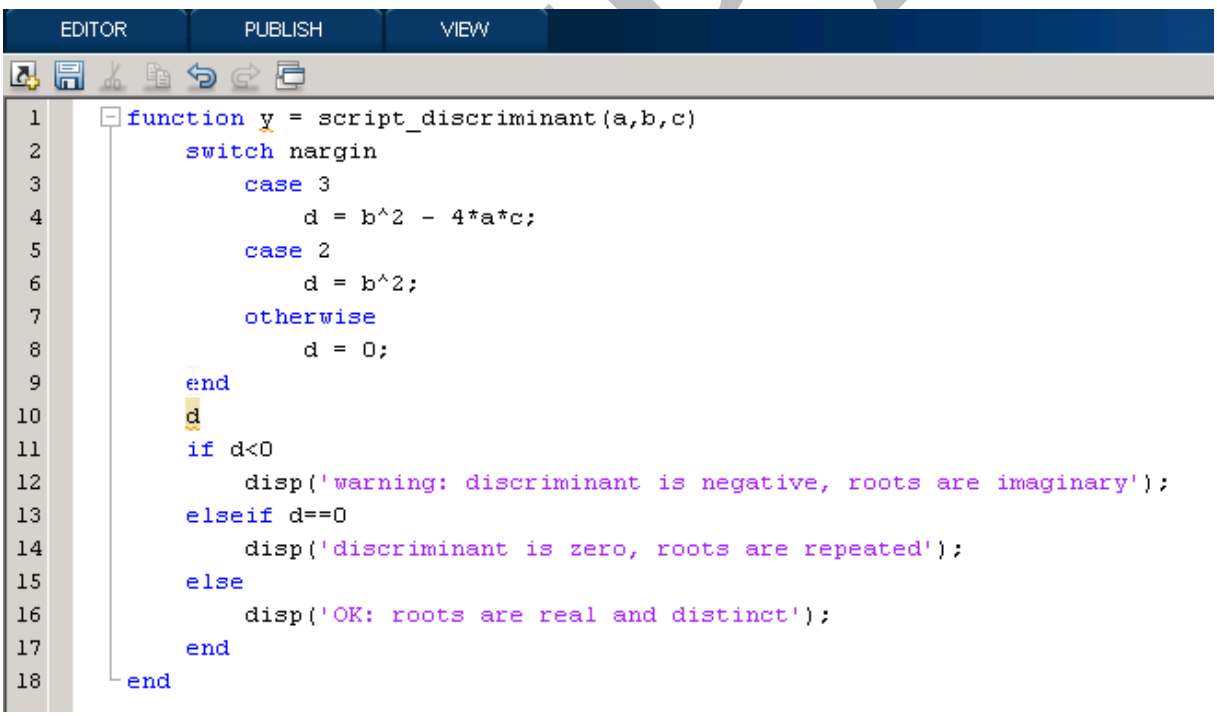
The **Switch...Case...Otherwise...End** statement is known to be faster as the If statement in most case and gives you also the possibility to have a more readable script.

Here is a first example that is a variation of the previous one use to introduce if**…** statement:

```
mat_Random=rand(1,1);
[r, c]=size(mat_Random);

switch true
    case r==1 && c==1
        disp('scalar');
    case r==1 || c==1
        disp('vector');
    otherwise
        disp('matrix');
end
```

Here is another very interesting example introducing at the same time the famous **nargin** MATLAB command:




```
function y = script_discriminant(a,b,c)
    switch nargin
        case 3
            d = b^2 - 4*a*c;
        case 2
            d = b^2;
        otherwise
            d = 0;
    end
    d
    if d<0
        disp('warning: discriminant is negative, roots are imaginary');
    elseif d==0
        disp('discriminant is zero, roots are repeated');
    else
        disp('OK: roots are real and distinct');
    end
end
```

As you can see above, mix **Switch** with **If** in the same function/script is not a major issue (logic...). The result of this script if you call the function will be:

```
Command Window
>> script_discriminant(5,2,3)

d =

   -56

warning: discriminant is negative, roots are imaginary
>>
```

Another typical very simple example applied to finance:

EDITOR PUBLISH VIEW

```
1  function [value] = diffbls2price(PutOrCall,S0,X,r,sigma,T,div,Optionprice)
2      %diffbls2price function
3      % dist computes the difference between the function
4      % BlackScholes option value and the given option price.
5      [call, put] = blsprice(S0,X,r,T,sigma,div);
6      switch PutOrCall
7          case 'Call'
8              value = call - Optionprice;
9          case 'Put'
10             value = put - optionprice;
11         otherwise
12             error('Check option type!');
13     end
14 end
```

SAMPLE

Iteration structures

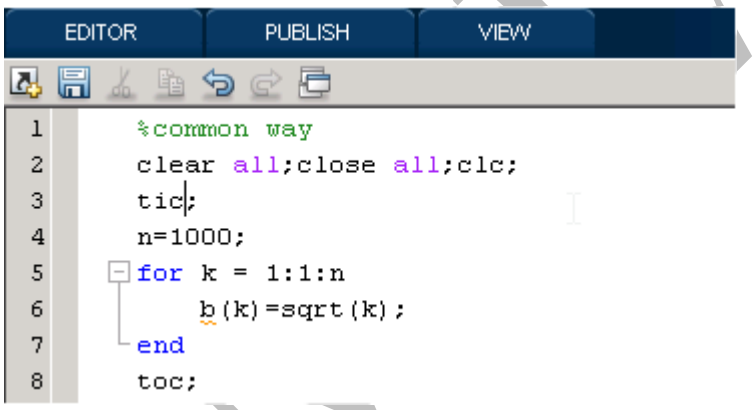
Iteration structures are well known in algorithmic and thus in any programming language. There is just a trap with MATLAB: **even if some MATLAB native functions facilitates the cleaning/treatment of vectors and matrices, sometimes the traditional For...End are faster if you test them with a tic... toc!**

You must also know that there is no **For each...** iteration structure in MATLAB (at least actually) and thus you cannot iterate across objects collection.

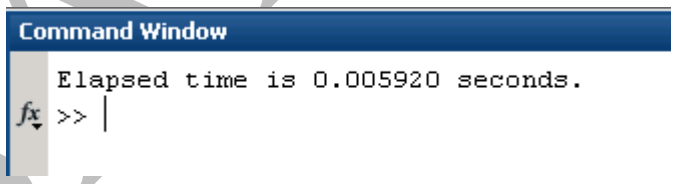
For...End iteration structure (and optimization)

In MATLAB we almost always use loop as in other programming/script languages. But the fact that MATLAB is Matrix based sometimes changes the deal with classic programming/script languages.

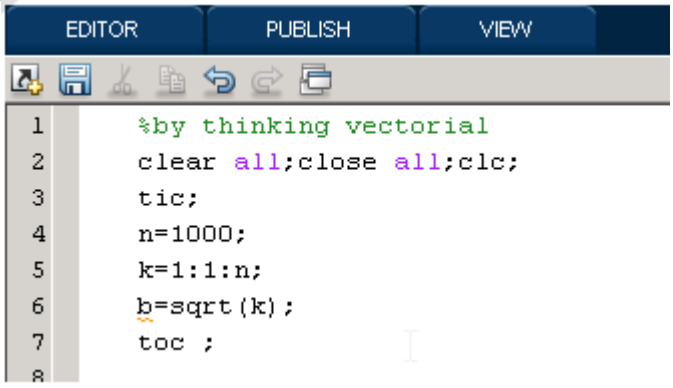
Let us see below a very easy example that shows us this paradigm:



and if we run the script (**Ctrl+Enter**) we get:

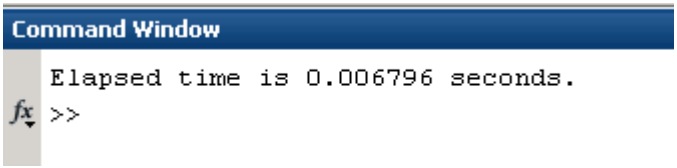


Now if we write the same thing but in a "vectorial" way we have:



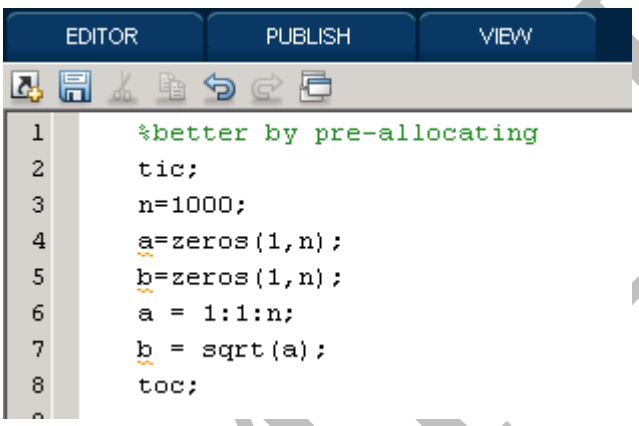
SAMPLE

and if we run the script (**Ctrl+Enter**) we get:

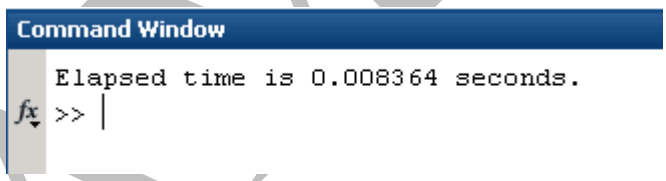


Take care to the statistical bias of the interpretation of only one run! You have to run a code a thousand of times to have a statistical distribution or the running time and after do a Student-*t* test (or a non-parametric median test) to compare the efficiency of the code.

Now what happens if we preallocate the vectors=:

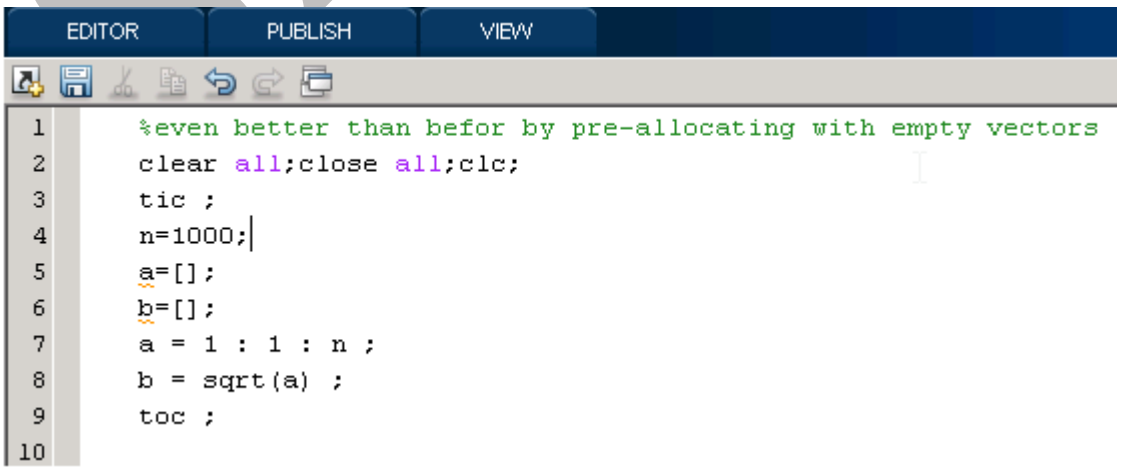


and if we run the script (**Ctrl+Enter**) we get:



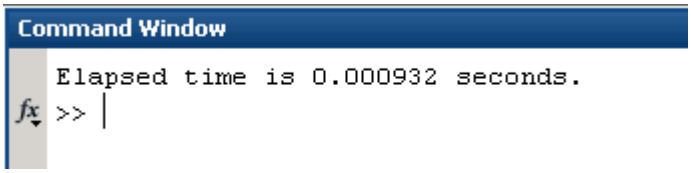
Seems to be much better...

And what happens if we create empty vectors only?:



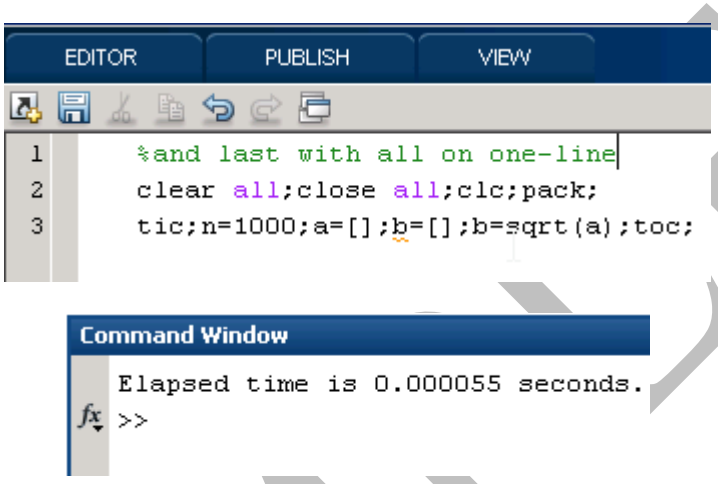
SAMPLE

and if we run the script (**Ctrl+Enter**) we get:

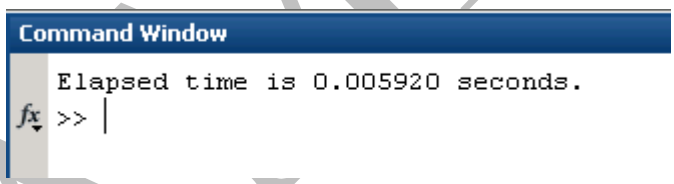


wouah! Not too bad...

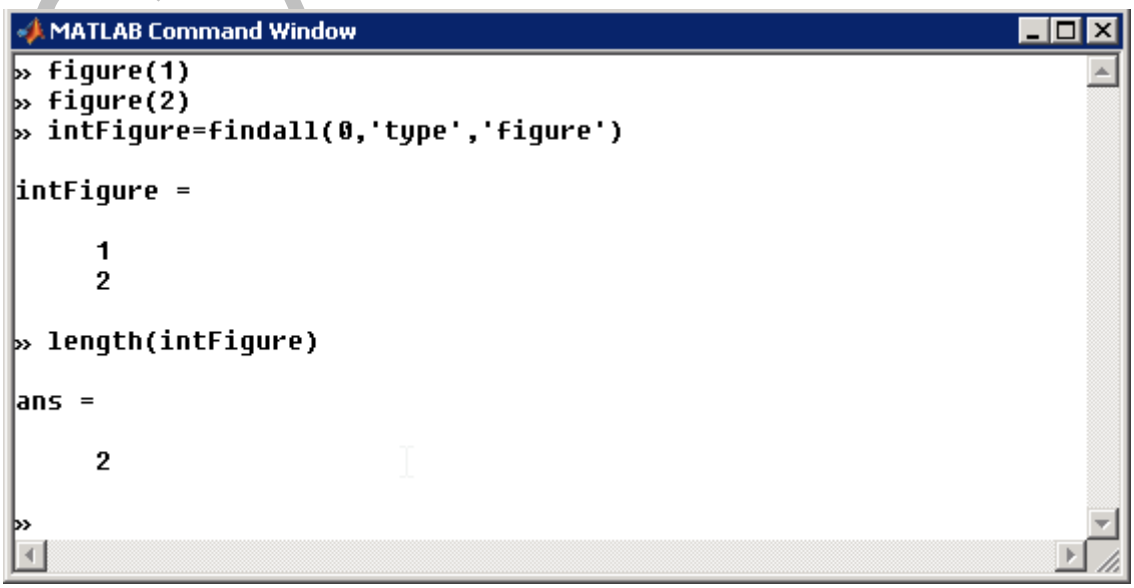
And what happens if we write a maximum of stuff only on one line?:



wouhahah! In comparison with the first script we are **1000 times faster**!

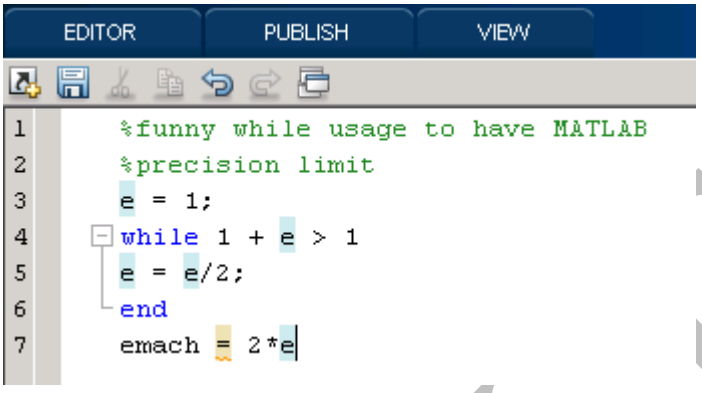


A last remark... If you want to loop on chart figures you will sometimes need to know how much figure are open. For this purpose you can use the following syntax:

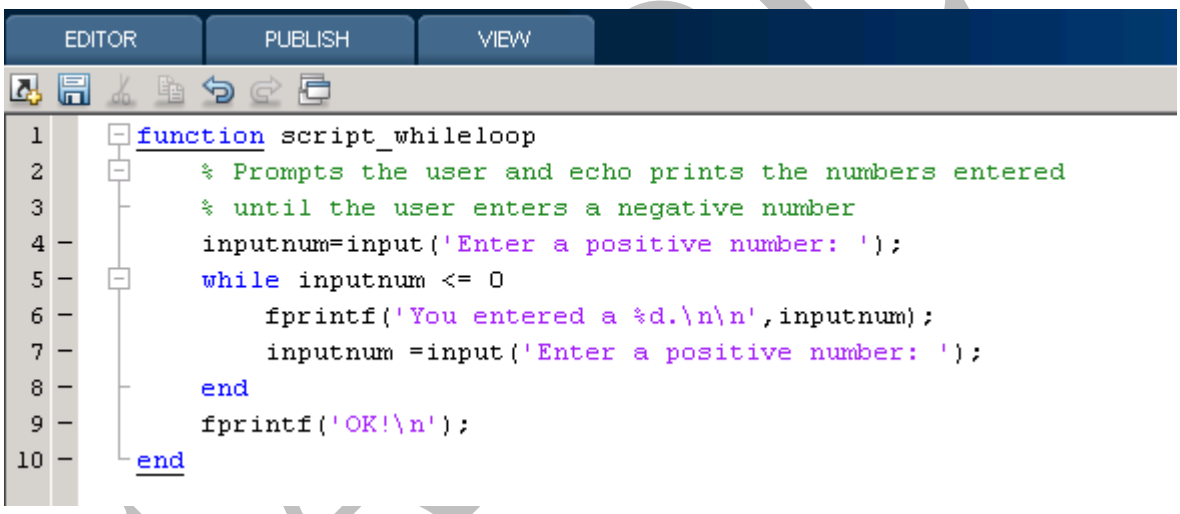


While...End iteration structure

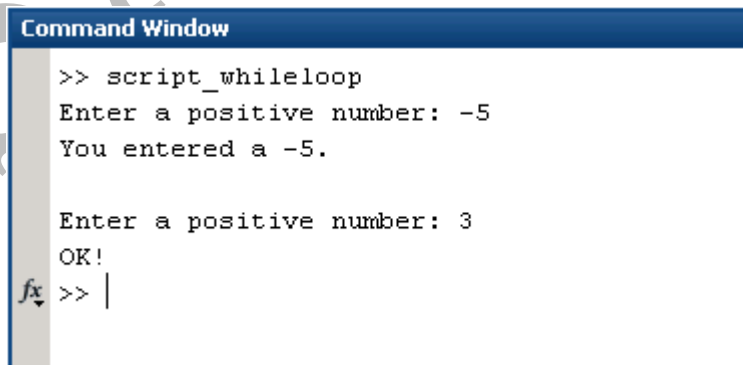
For...End iteration structures are not always adapted to all situations. Before that we see such a real example let us see the syntax of a **While...End** structure with a funny well know example of the precision limit on any programming/script language:



Or the very common case when you don't have developed a GUI (Graphic User Interfacce):



If we call the function we get something working well:



Another typical example using a **while** with multiple conditions:

```
clear all;clc;

r=-1;c=-1;
while (r<=0 || c<=0)
    clc
    r=input('Type the number of desired rows:');
    c=input('Type the number of desired columns:');
end

mat_Random=rand(r,c);
[r, c]=size(mat_Random);

switch true
    case r==1 && c==1
        disp('scalar');
    case r==1 || c==1
        disp('vector');
        for k=1:max([r,c])
            disp(mat_Random(k));
        end
    otherwise
        disp('matrix');
end
```

SAMPLE

Functions (functions files)

In some example seen before you have certainly noted that when you create a function, t**he *.m file must have the same name as the function itself**!

You can call MATLAB functions using either **command syntax or function syntax**, as described below.

A function call in this syntax consists of the function name followed by one or more arguments separated by spaces:

functionname arg1 arg2 ... argn

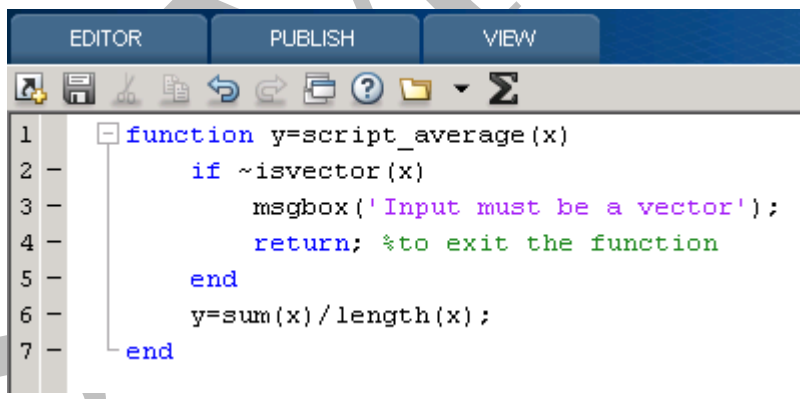
Command syntax does not allow you to obtain any values that might be returned by the function. Attempting to assign output from the function to a variable using command syntax generates an error. Use function syntax instead.

A function call in this syntax consists of the function name followed by one or more arguments separated by commas and enclosed in parentheses:

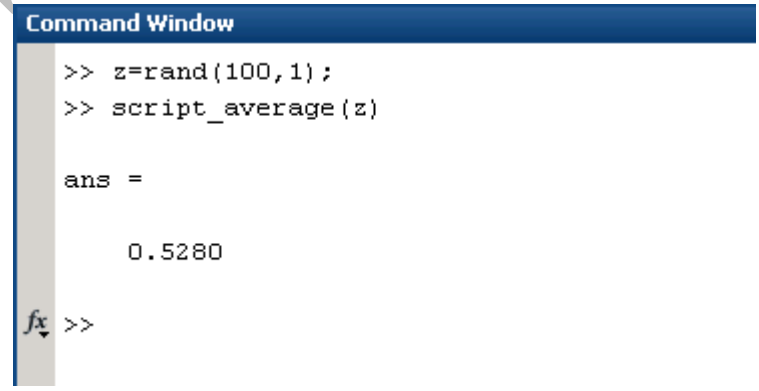
functionname(arg1, arg2, ..., argn)

One input, One output (OIOO)

Now write one of the simplest example that you can imagine with a function: something that takes in input only one information and that returns also only one output:

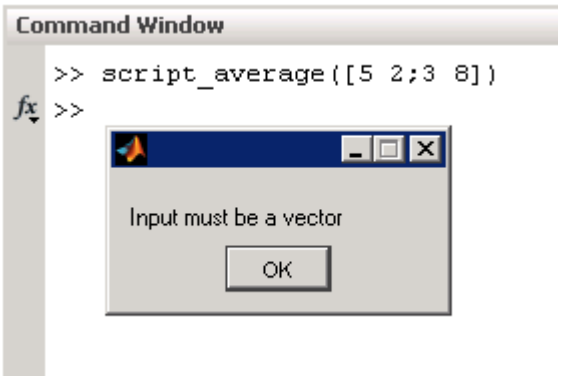


If you call the function you simply get:



SAMPLE

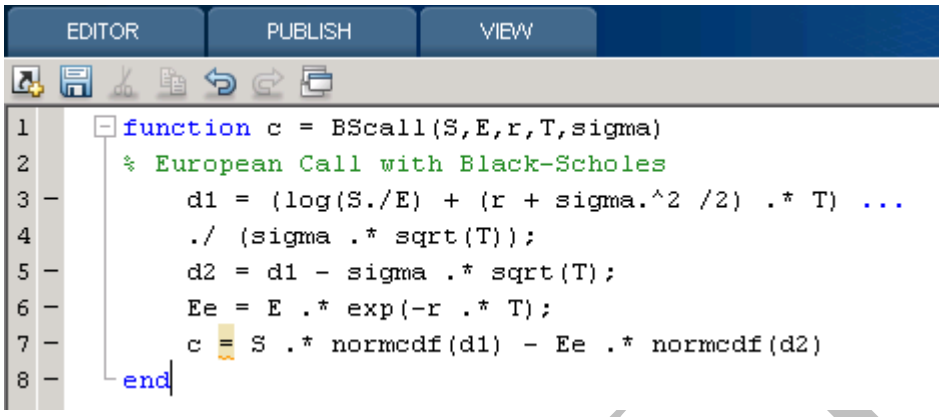
Nothing new! And if generate an error:



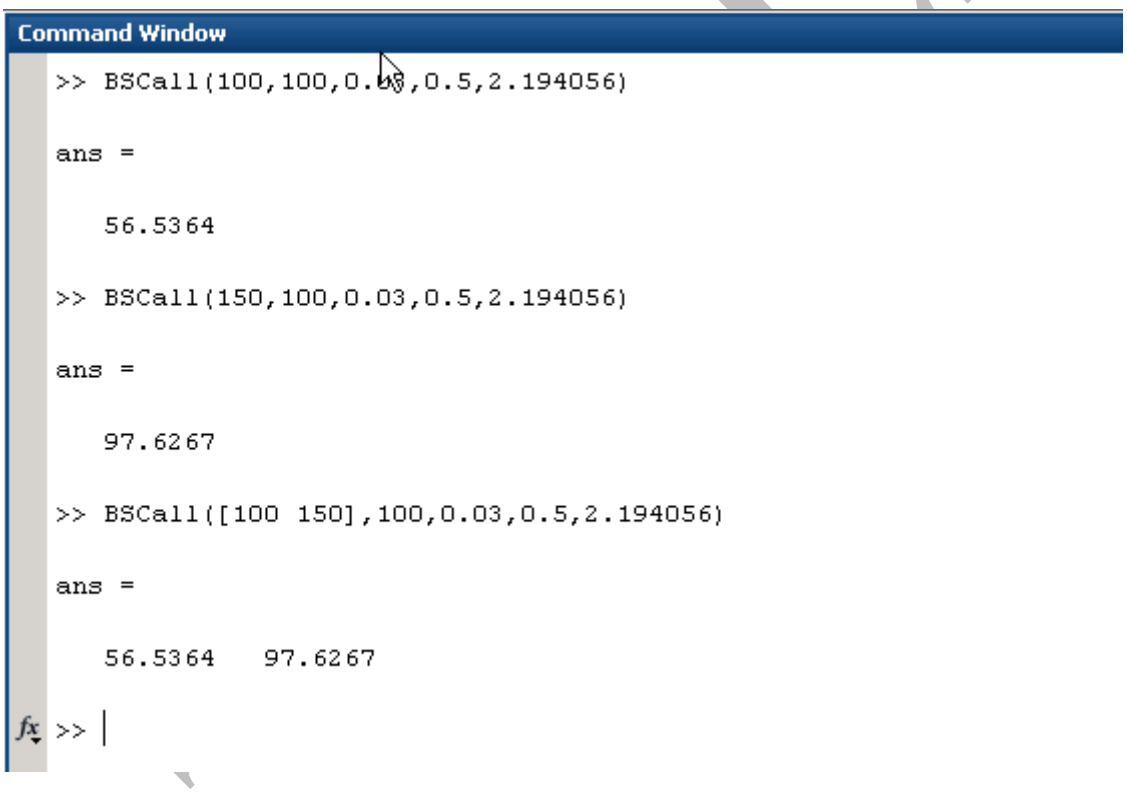
SAMPLE

Multiple inputs, One output (MIOO)

The level just above of the previous example is to have multiple inputs and to return one output. Here is also a well-known worldwide example in Finance of such a situation:



So we get:



As you can see, the inputs can also be vectors!

One input, Multiple outputs (OIMO)

This is also a famous example introducing the **nargout** MATLAB command that detects how many outputs the user request for the function:

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\stats.m

EDITOR PUBLISH VIEW

```
1   function [mu, sigma]=stats(X)
2       %statistics will be calculate only if there are two requested outputs
3       if nargout==2
4           mu=mean(X);
5           [nobs,nc]=size(X);
6           sigma=sqrt(sum(X.^2)/(nobs*nc)-mu.^2);
7       else
8           disp('You must request two outputs!');
9       end
10  end
```

The result will be:

Command Window

```
>> v=[5 7 2];
>> stats(v)
You must request two outputs!
>> [mu,sigma]=stats(v)

mu =

    4.6667


sigma =

    2.0548

>> [~,sigma]=stats(v)

sigma =

    2.0548

fx >>
```

And if we simplify the code:

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\stats.m

EDITOR PUBLISH VIEW

```
1   function [mu, sigma]=stats(X)
2       mu=mean(X);
3       [nobs,nc]=size(X);
4       sigma=sqrt(sum(X.^2)/(nobs*nc)-mu.^2);
5   end
```

We get:

Command Window

```
>> v=[5 7 2];
>> stats(v)

ans =

    4.6667

>> [mu, sigma]=stats(v)

mu =

    4.6667


sigma =

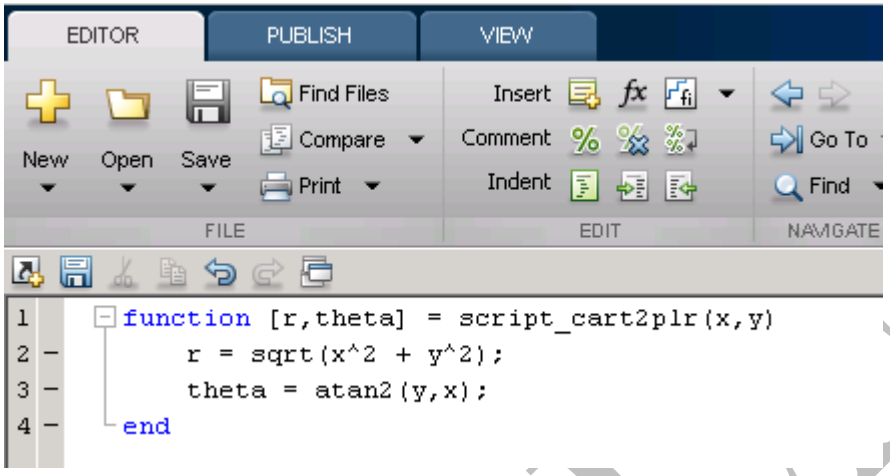
    2.0548

>>
```

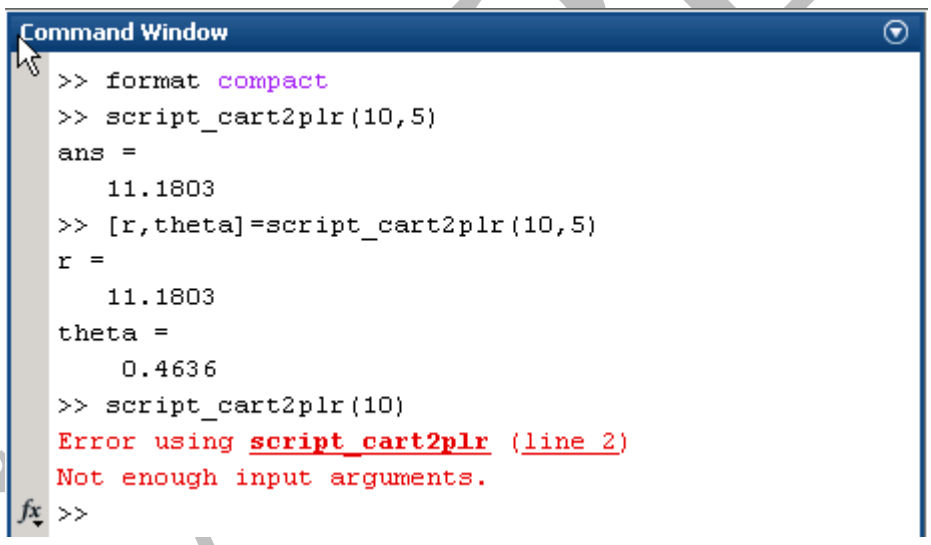
Nothing very suprising.

Multiple inputs, Multiple outputs (MIMO)

The way to use multiple inputs to get multiple output is by extension very easy. Consider the following well know academic example:

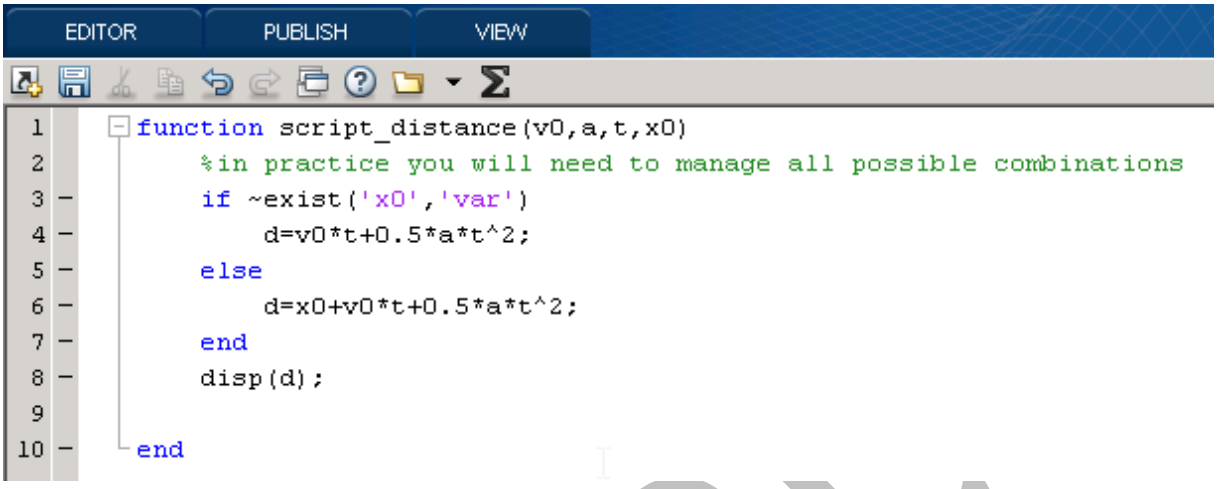


The result will be:

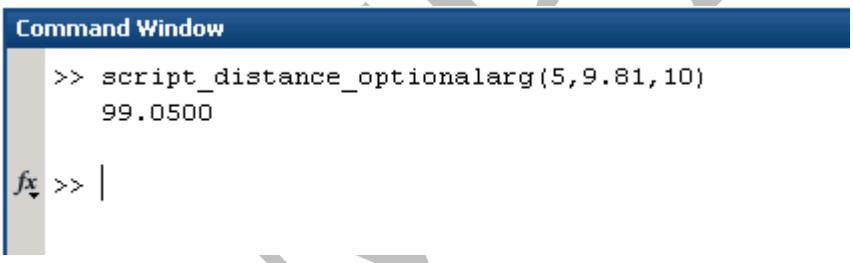


Optional arguments

We saw before that we can use **nargin** and **nargout** to manage/detect respectively input and output arguments. If you don't like how this both functions work you can use instead the function **exist( )**:



As you can see, if we forget the fourth argument the function still work:



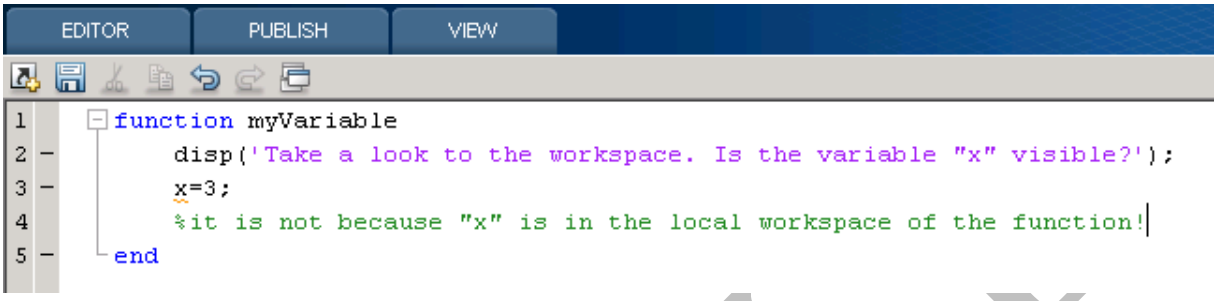
SAMPLE

Variables scope

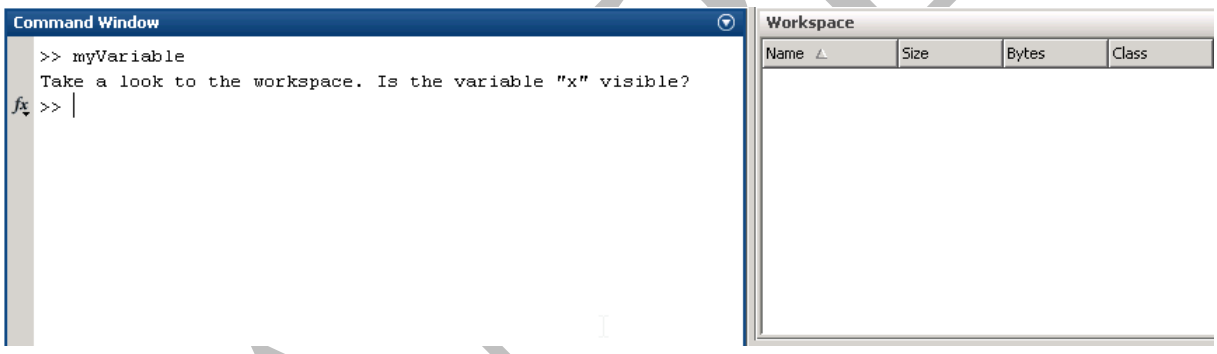
Most variables in the majority of script are local but in some special case it is useful to manage the scopes of variables.

Global vs Local variables

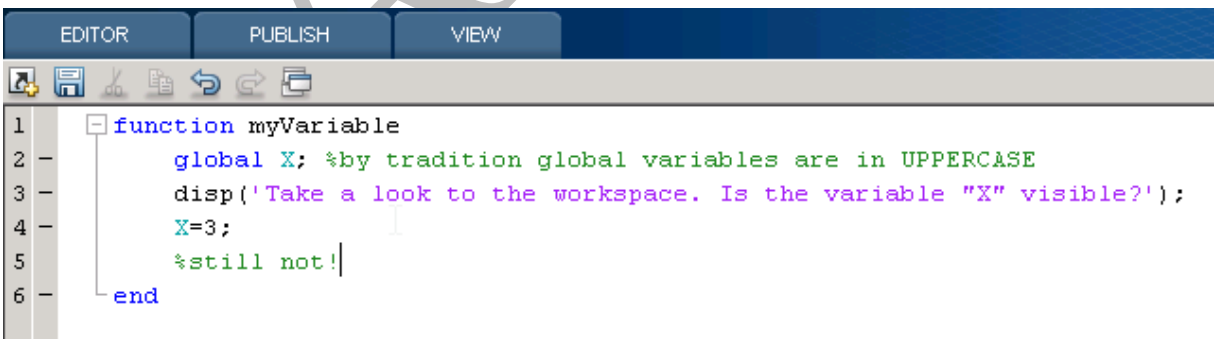
The first example is to see what happens when you create a variable into a function:



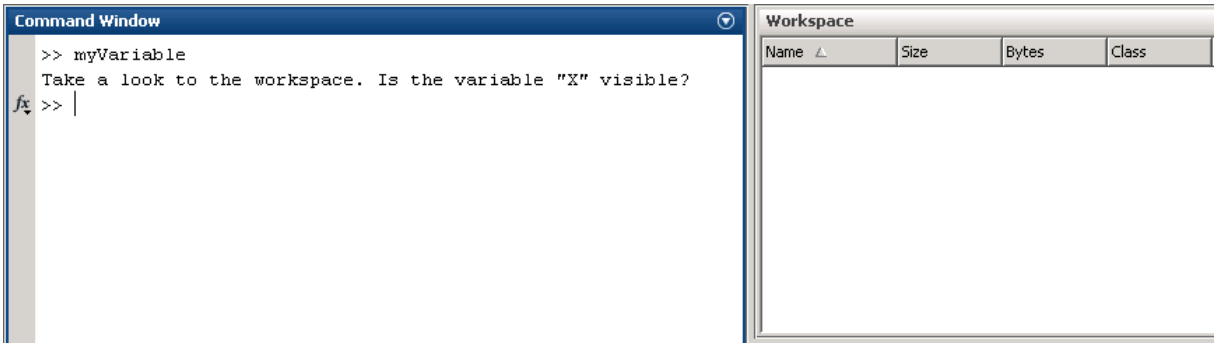
You will see that the variable is hopefully by default not visible in the MATLAB workspace. This is why we call such a variable a "ultra-local variable":



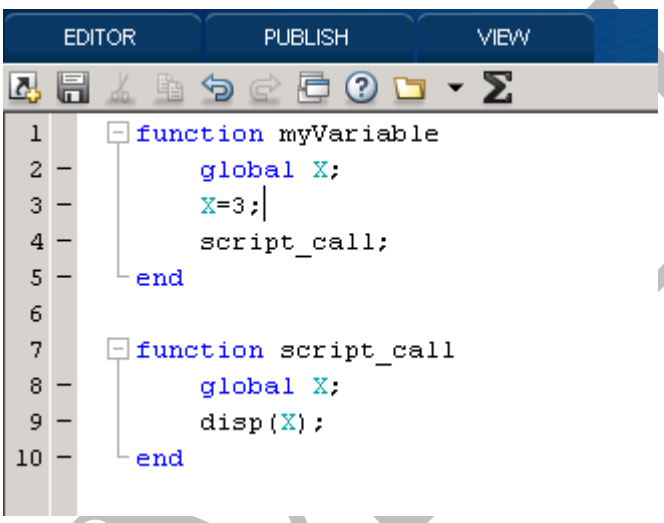
Now see what happens if we declrate the variable using the statement **global**:



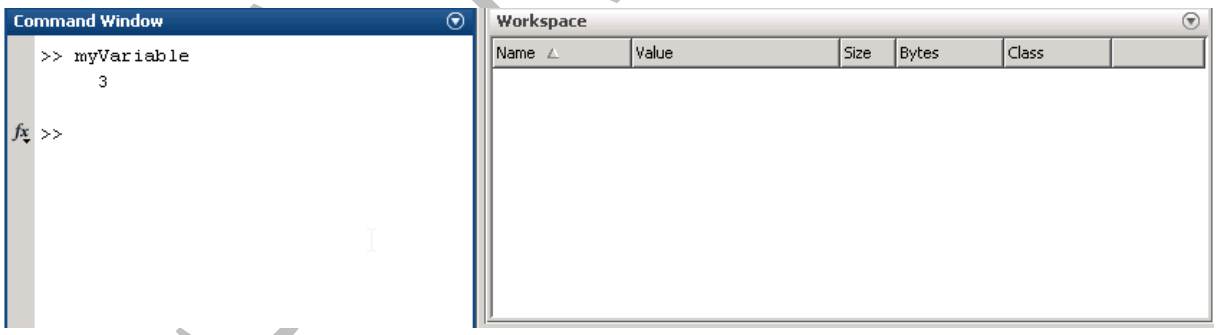
As you can see the variable is still (hopefully) not visible in the MATLAB workspace:



Then what is the purpose? The purpose is that the variable value can still be accessed by another function:



And we get:

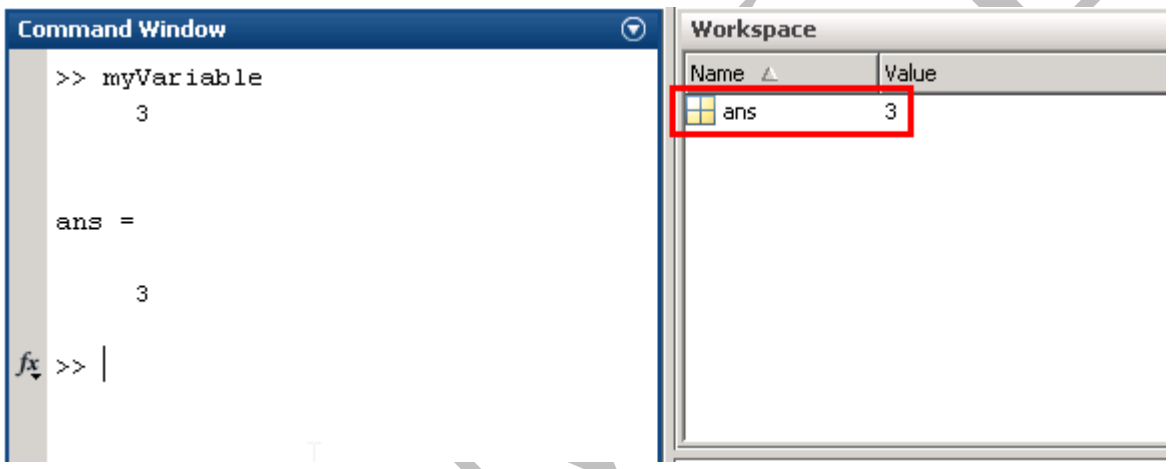


Then, what we call a "global variable" is a variable that is global at the level of functions workspace but will by default never be visible in the MATLAB workspace for security purposes (to not erase variables having the same name).

If you want you can change the script as following:

```
1   function X=myVariable
2       global X;
3       X=3;
4       script_call;
5   end
6
7   function script_call
8       global X;
9       disp(X);
10  end
```

Then when you run it you will get:



SAMPLE

Persistent variables

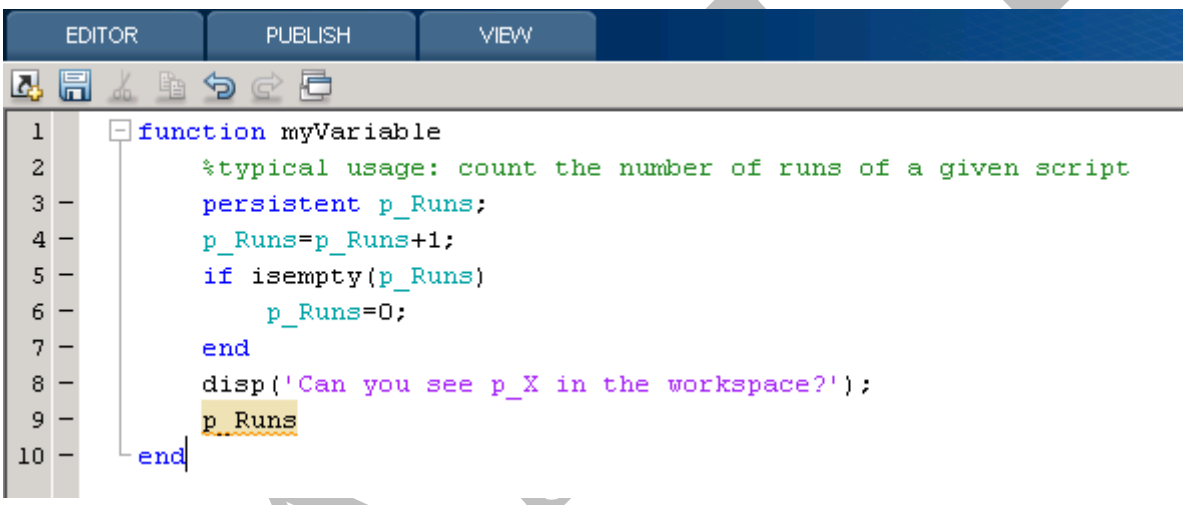
Some comments are in order:

1. You can declare and use persistent variables in a M-file function only (this is on of the main difference with a Global Variable that can be modified by the user in the MATLAB GUI).

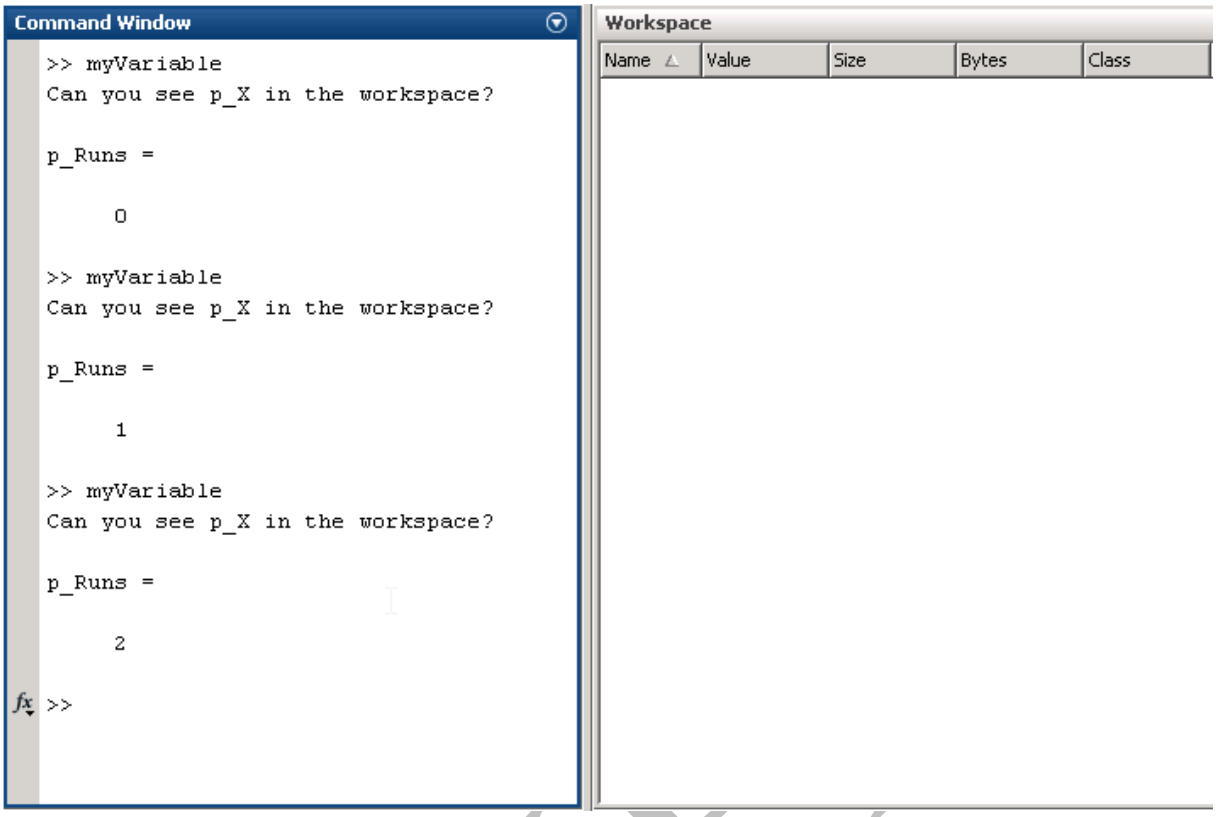
2. Only the function in which the persistent variable is declared is allowed access to the variable (this is specific to persistent variables). At the opposite of a Global variable that is Global AND persistent

3. When the function exits, MATLAB does not clear the persistent variable from memory, so it retains its value from one function call to the next (this is the same behavior as global variables)

To declare a persistent variable we use the statement **persistent**:



SAMPLE

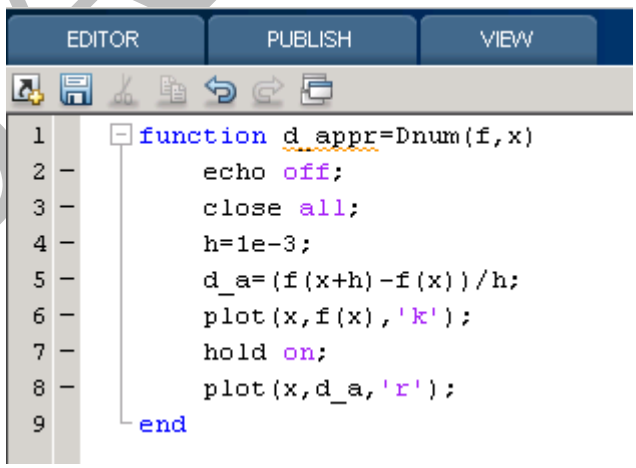


As you can see below, the variable is never visible in the MATLAB workspace!

Anonymous function as input, chart output

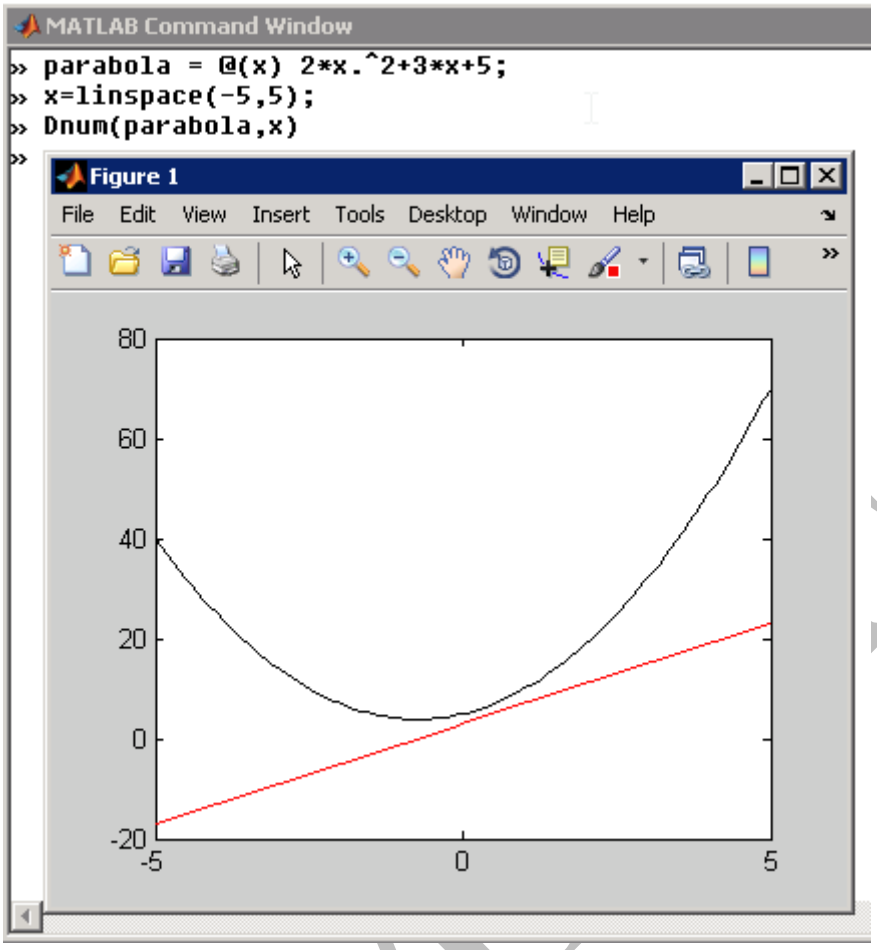
One very interesting and important thing to know is that you can also pass functions as argument.

Consider the following simple example that just calculate the derivate of any function in a range defined by an input vector:



Then we can call this function as following using the anonymous **@** function statement:

SAMPLE



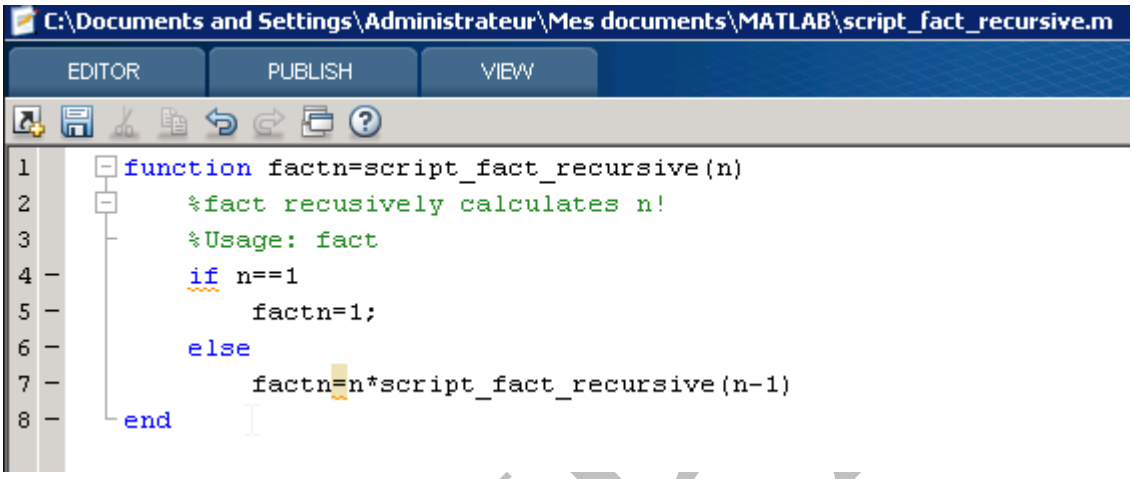
Works well :-)

SAMPLE

Recursive functions

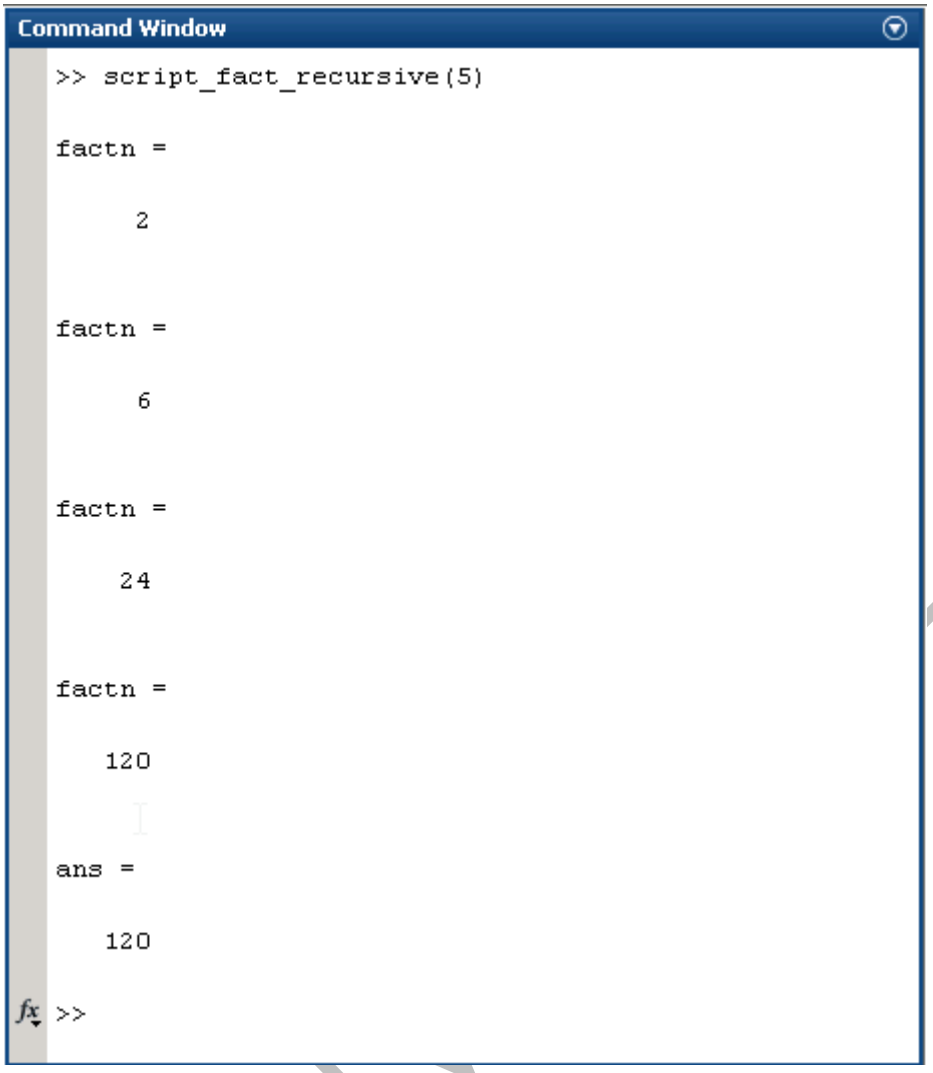
Another useful type of function (especially in finance for example with binomial/trinomial trees and Data Mining for hierarchical clustering) are recursive function that call themselves to compute the next values.

As example see the famous academic example below where we calculate the factorial using recursivity:



Thus we get:

SAMP

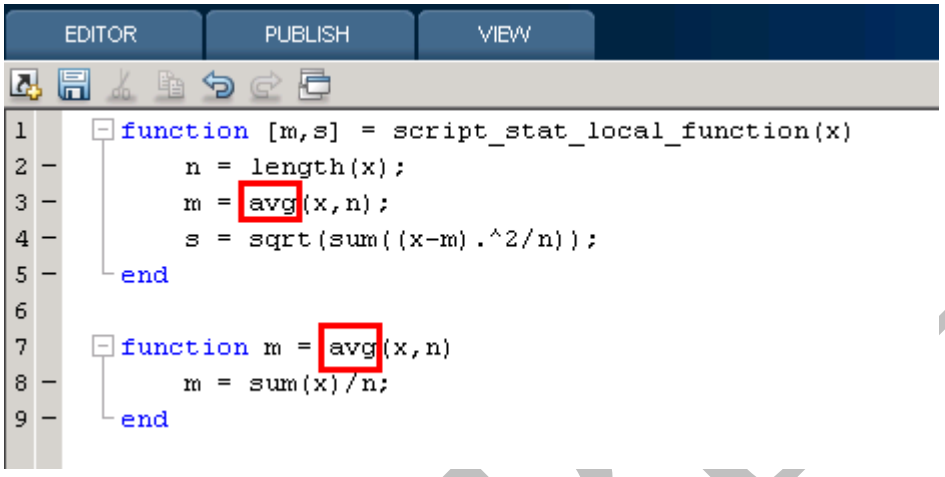

Command Window
>> script_fact_recursive(5)
factn =
2
factn =
6
factn =
24
factn =
120
ans =
120
>>

SAMPLE

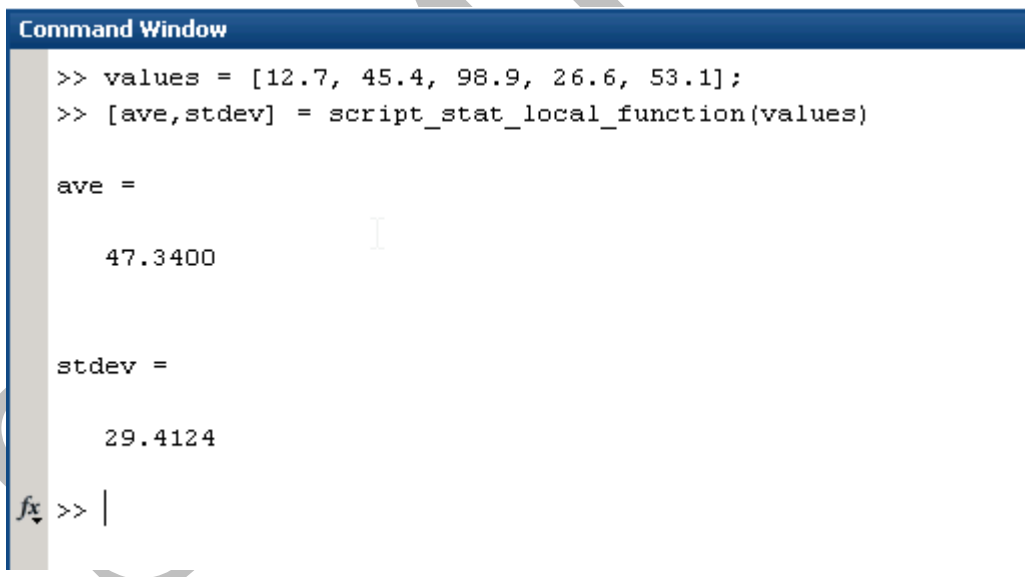
Local functions (multiple functions in one script)

As in many scripting/programming languages it must be possible in a same M-file to have multiple functions.

As you can see this works well but the only one condition is that the main callable function to be at the top of the script:



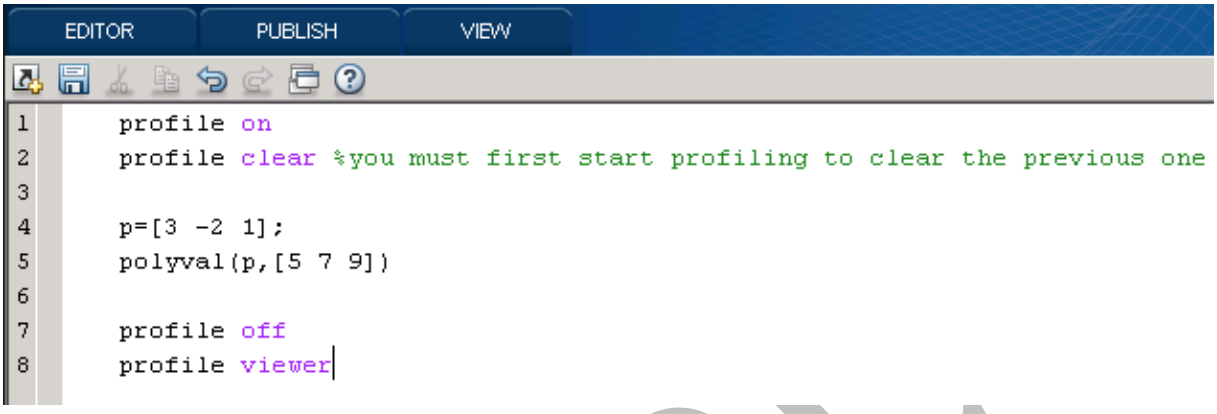
As you can see this works well:



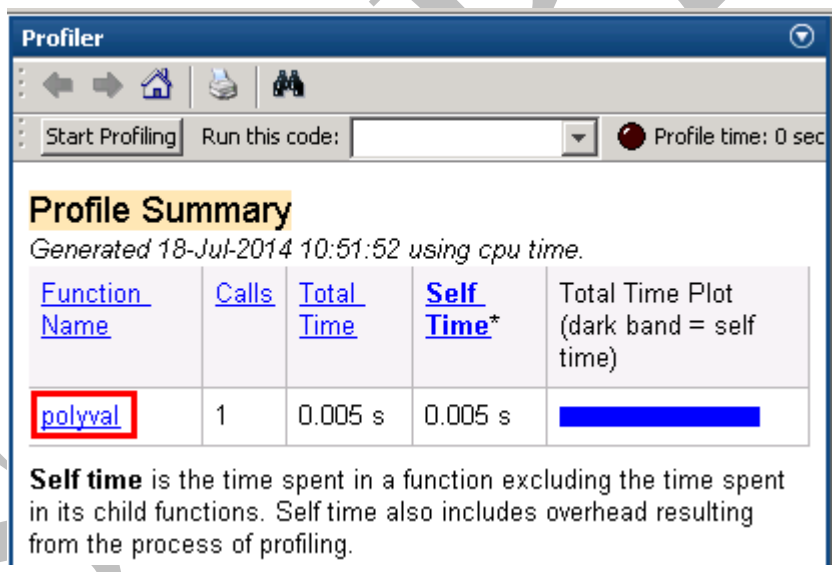
Profile a script

Profiling a script is a very useful functionality when you want to see where you code has low performances pieces.

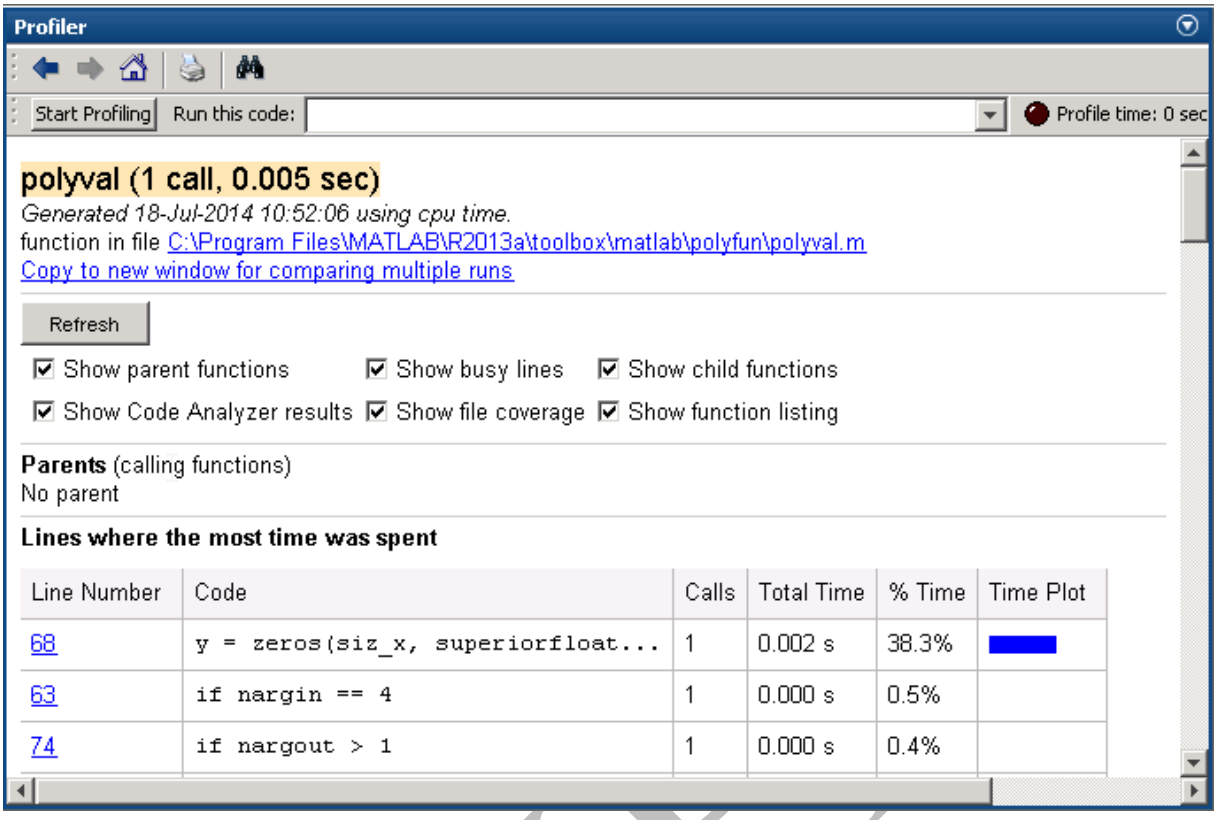
To use this functionality we will need the commands **profile on**, **profile clear**, **profile off** and finally **profile viewer**:



Then you will see a new panel that evaluates the execution time of the function **polyval( )**:



If you click on the polyval link you get more details about what happens inside this function:



Simple but very useful!

As far as I know you can't use directly profiling command directly in function. Thus if you consider a function like the one below (empirical example):

script_to_profile.m | script_profile_test.m

```
%% This is a demo script with typical help
% Script Autocompletion shows an autocompletion demo
%   script_autocompletion makes a simple calculation and returns only
%
%   See also SUM, PLUS.
%
%   For more information, see <a href="matlab:web('http://www.mathwor
%
%   Copyright 2015 Mathworks

%   $Author: V. ISOZ $  $Date: 2015/05/29 18:32:43 $    $Revision: 0.

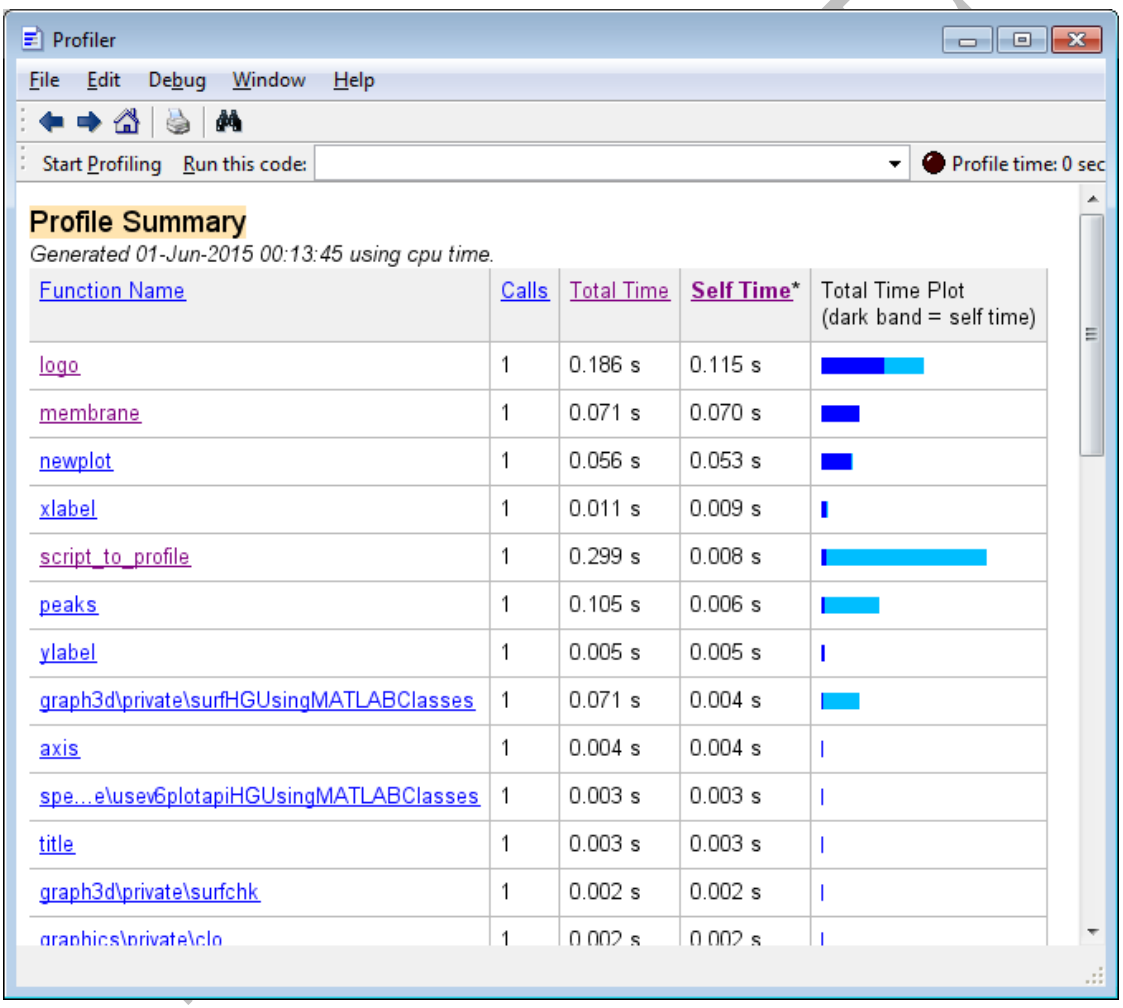
%ToDo: Ne pas oublier de décommenter la fonction
function y=script_to_profile(x)
    echo on;
    %ToDo: Ne pas oublier d'utiliser la variable
    n=10000; %#ok<NASGU>
    if ~isvector(x)
        %FixMe: Ne pas oublier de faire de la vraie gestion d'erreurs
        errordlg('Input must be vector');
    end
    y=sum(x)/length(x);
    peaks;
    logo;
    %Note: Ceci reste à être validé par le client
end
```

To test this function the only one solution I know for the moment is to write in another script:

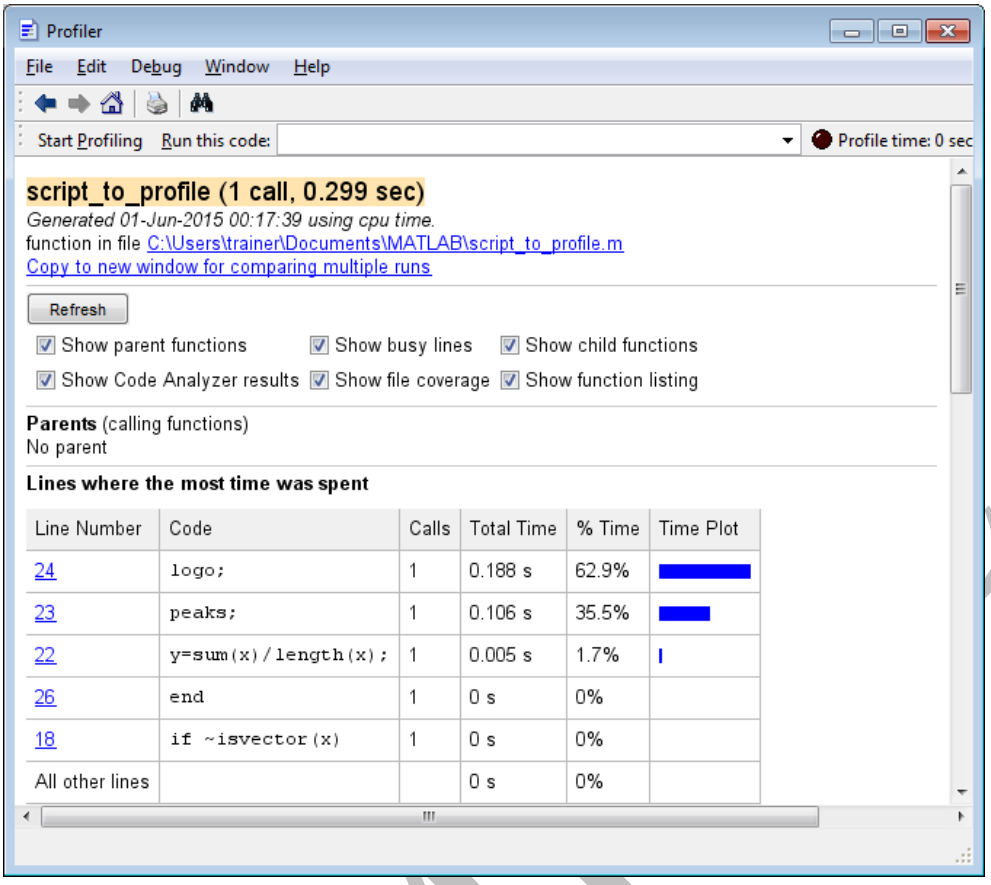
script_to_profile.m | script_profile_test.m

```
1 -    profile clear
2 -    profile on
3
4 -    script_to_profile(rand(1e7,1));
5
6 -    profile off
7 -    profile viewer
```

So we get:



and if you click on the function *script_to_file* you get:

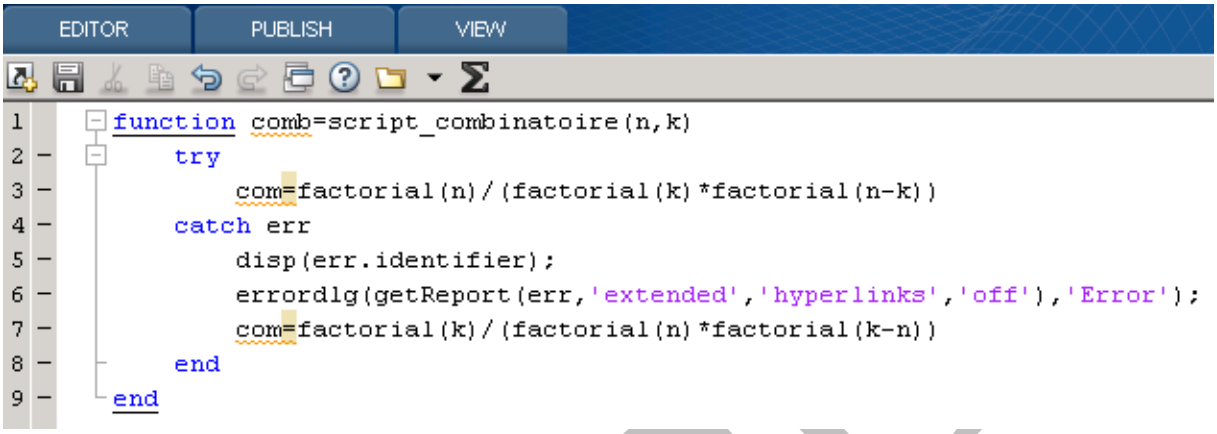

Profiler
File Edit Debug Window Help
Start Profiling Run this code:
Profile time: 0 sec
script_to_profile (1 call, 0.299 sec)
Generated 01-Jun-2015 00:17:39 using cpu time.
function in file C:\Users\trainer\Documents\MATLAB\script_to_profile.m
Copy to new window for comparing multiple runs
Refresh
Show parent functions
Show busy lines
Show child functions
Show Code Analyzer results
Show file coverage
Show function listing
Parents (calling functions)
No parent
Lines where the most time was spent
Line Number | Code | Calls | Total Time | % Time | Time Plot
24 | logo; | 1 | 0.188 s | 62.9%
23 | peaks; | 1 | 0.106 s | 35.5%
22 | y=sum(x)/length(x); | 1 | 0.005 s | 1.7%
26 | end | 1 | 0 s | 0%
18 | if ~isvector(x) | 1 | 0 s | 0%
All other lines | | | 0 s | 0%

SAMPLE

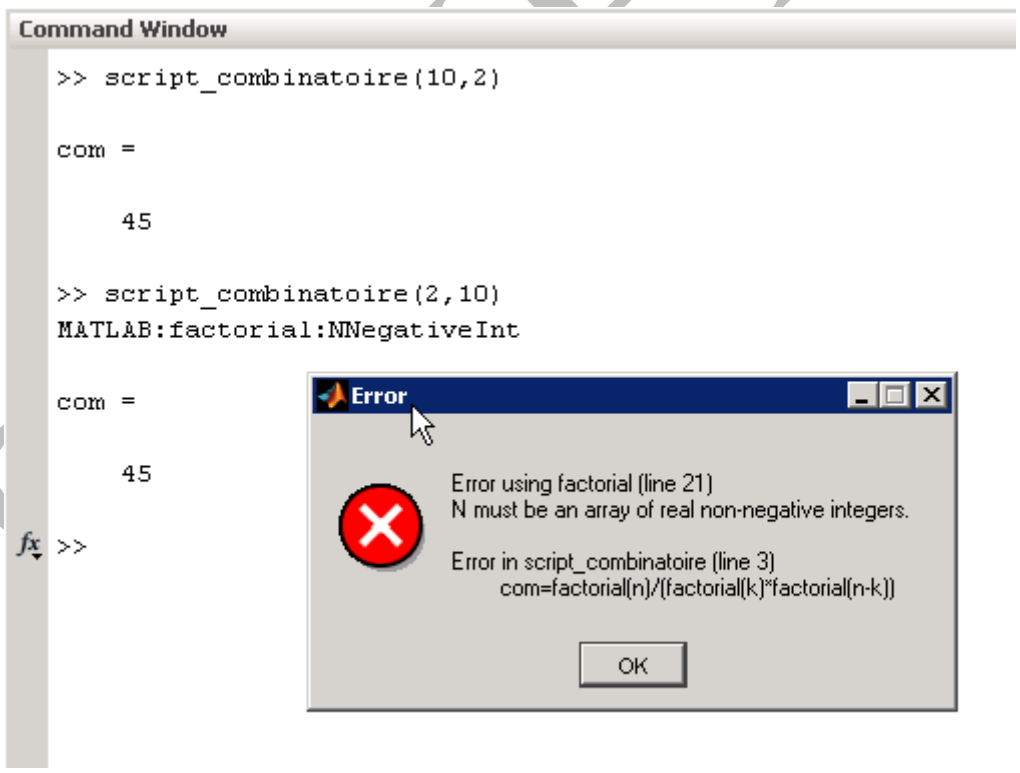
Catching errors

Catching errors is one of the most important job of a professional scripting/programming language developer.

The syntax of MATLAB for this purpose is similar to many modern programming languages using the **Try...Catch...** structure as you can see below in this simple example:



The result will looks like following:



*Multiple callable functions in one *.m file*

By default the only function accessible from outside is the first function whose name is the same as the script file name.

Here is an example of what we could to intuitively:

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\PCAfunctions.m

EDITOR PUBLISH VIEW

```
1   function parametricPCA()
2       disp('This is a paramtric Principal Component Analysis')
3   end
4
5   function nonparametricPCA()
6       disp('This is a non-paremtric rank based Principal Component Analysis')
7   end
8
9   function sPCA()
10      disp('This is a sparse Principal Component Analysis')
11  end
```

But the result will not be the expected one:

Command Window

```
>> parametricPCA
Undefined function or variable 'parametricPCA'.

>> nonparametricPCA
Undefined function or variable 'nonparametricPCA'.

>> sPCA
Undefined function or variable 'sPCA'.

Did you mean:
>> spa
```

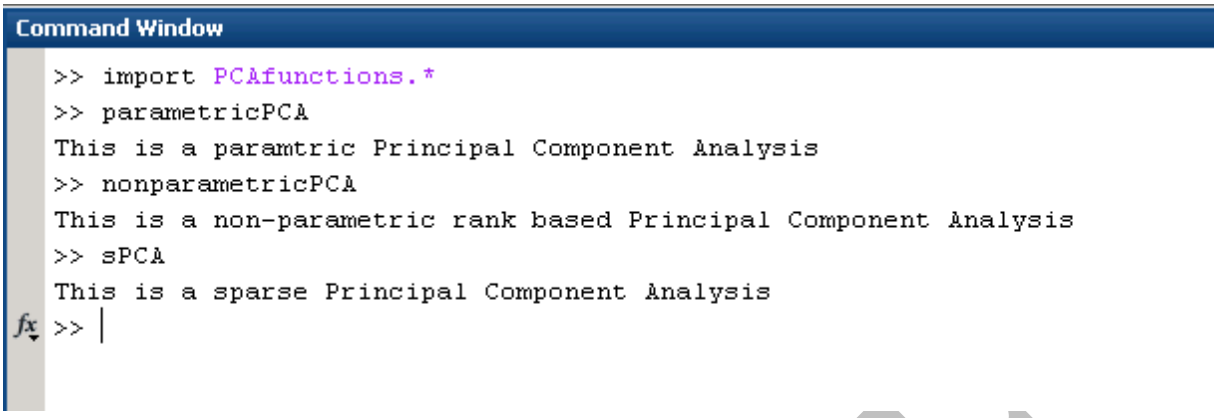
To be able to have multiple functions in a same file you have to use classes a following:

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\PCAfunctions.m

EDITOR PUBLISH VIEW

```
1   classdef PCAfunctions % in PCAfunctions.m
2       methods (Static = true)
3           function parametricPCA()
4               disp('This is a parametric Principal Component Analysis')
5           end
6
7           function nonparametricPCA()
8               disp('This is a non-parametric rank based Principal Component Analysis')
9           end
10
11          function sPCA()
12              disp('This is a sparse Principal Component Analysis')
13          end
14      end
15  end
```

After in any script or in the *Command Window* you just write:



At the same time this is a first introduction to OOP (Object Oriented Programming) in MATLAB.

SAMPLE

Debugging technics

Debugging is an important part of the job of any computer scientist. As we know one of the axioms of computing science is that almost all scripts have a least one potential bug (that is not necessary due to the code itself!)

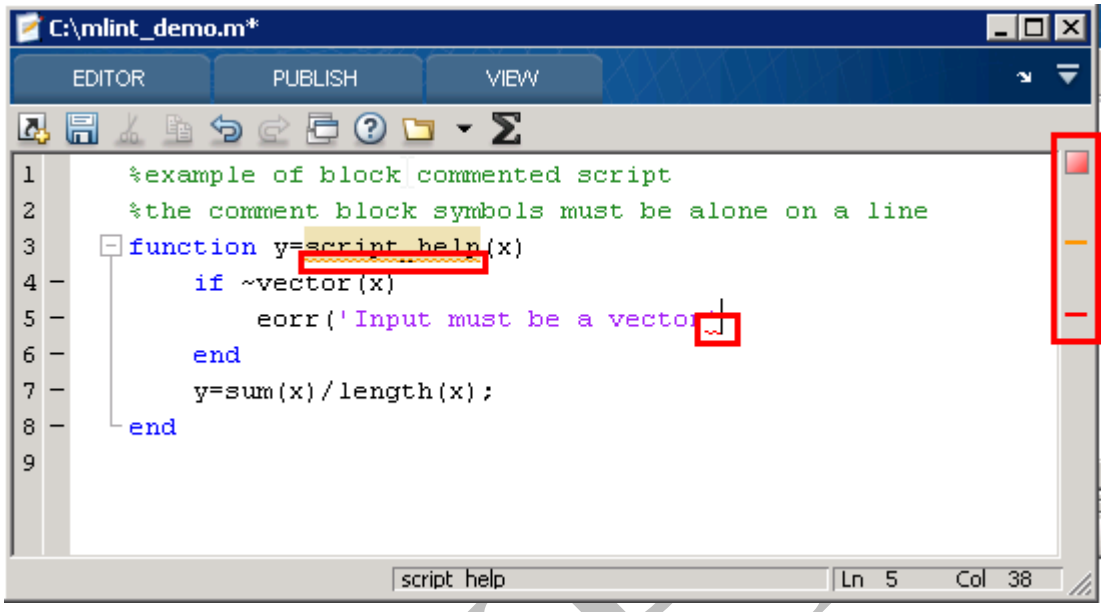
There are different commands for this purpose in MATLAB and let us see the most important of them.

SAMPLE

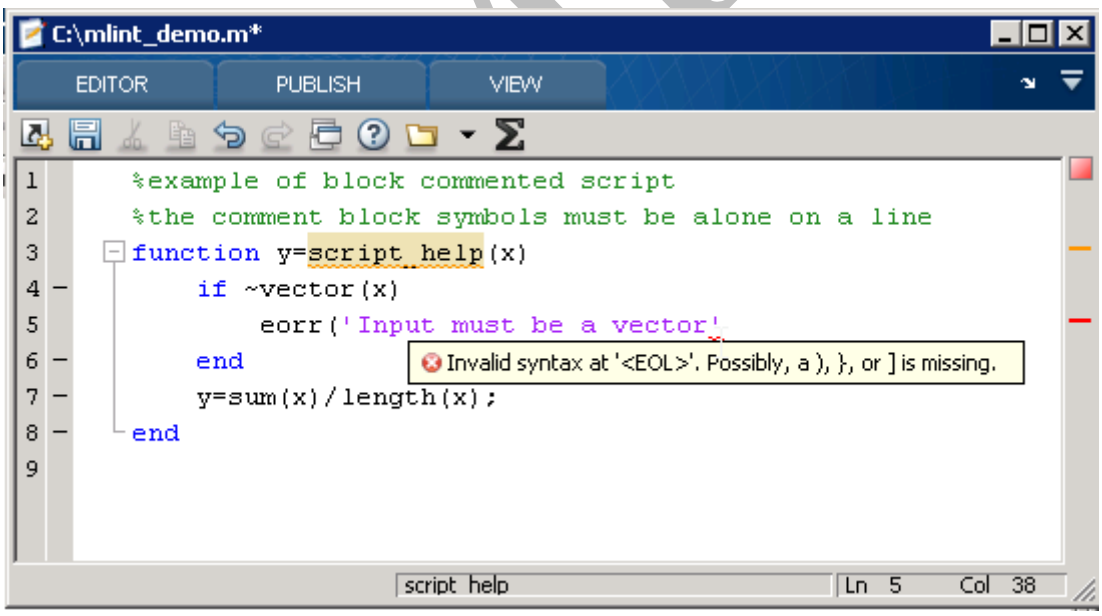
M-lint

M-lint real time analyzer

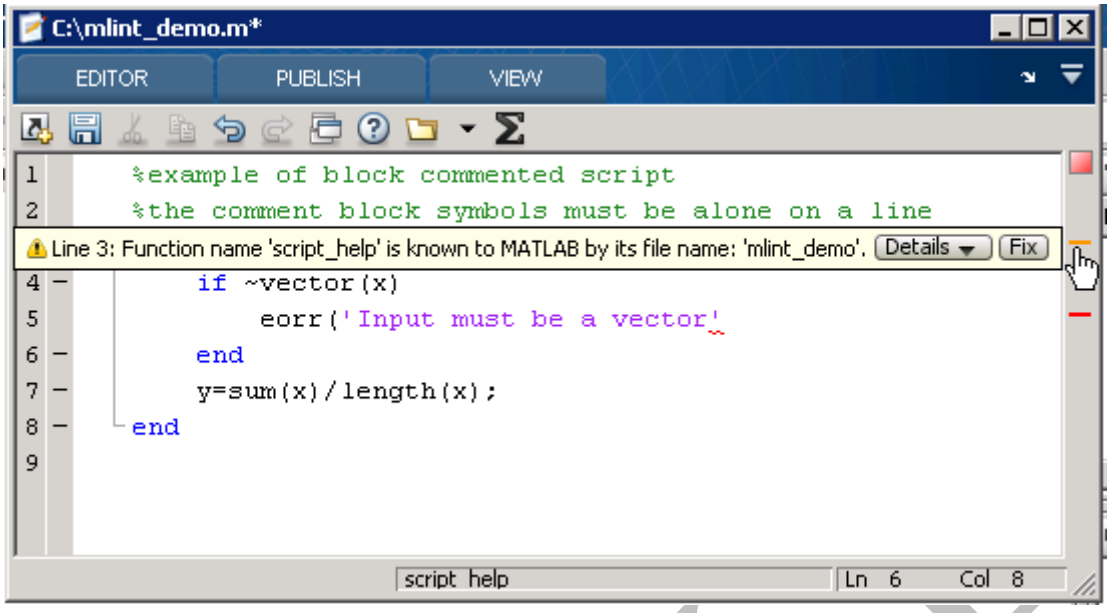
The M-lint real time analyzer is easy to recognize. Consider the following function and see:



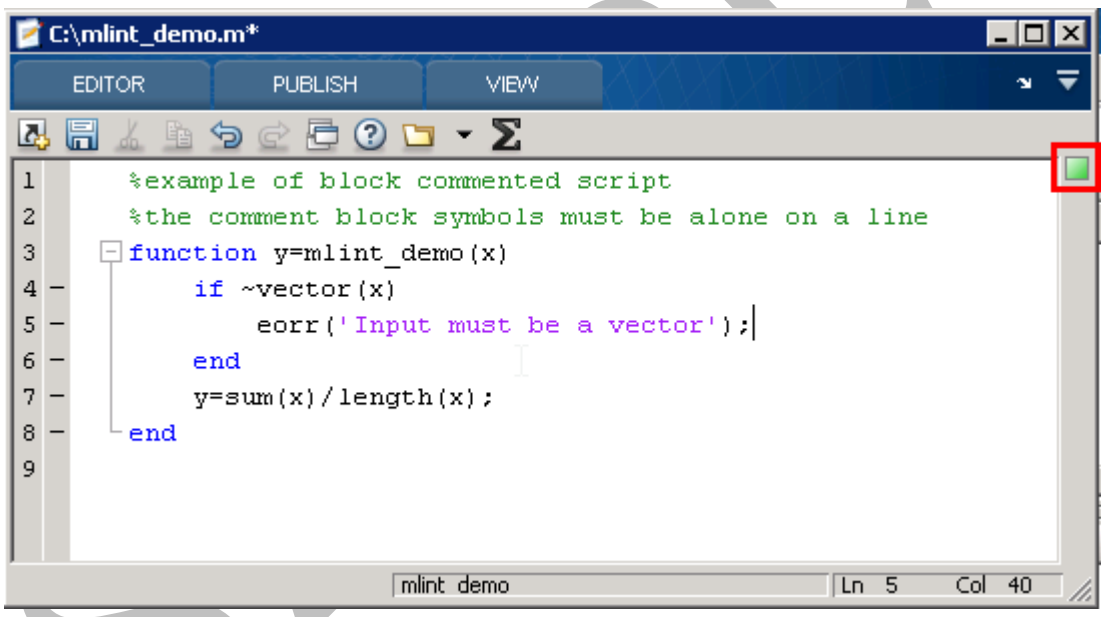
Going with the mouse on the colors we get for example:



Or:

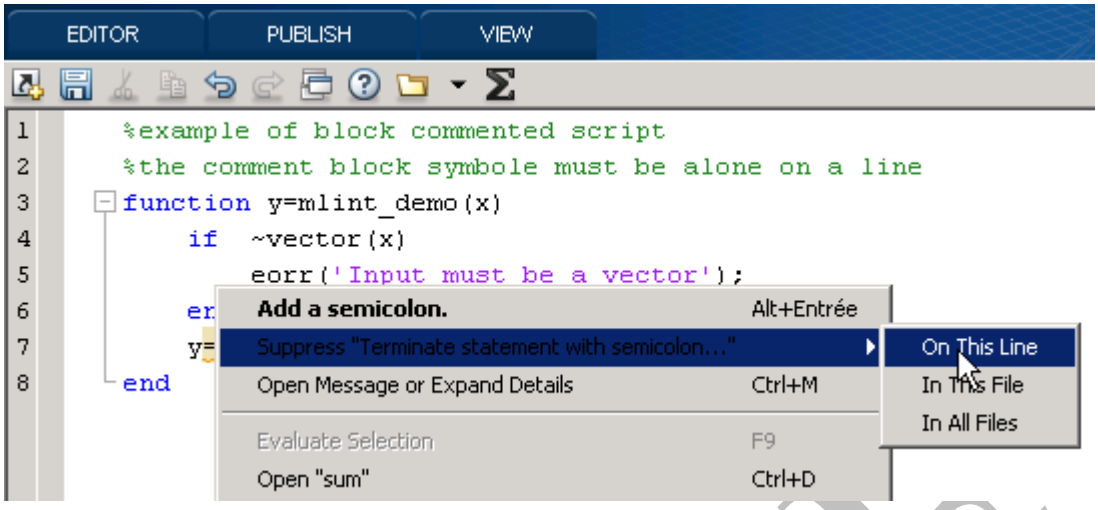


If the system thinks... that there are no errors you get a green square:

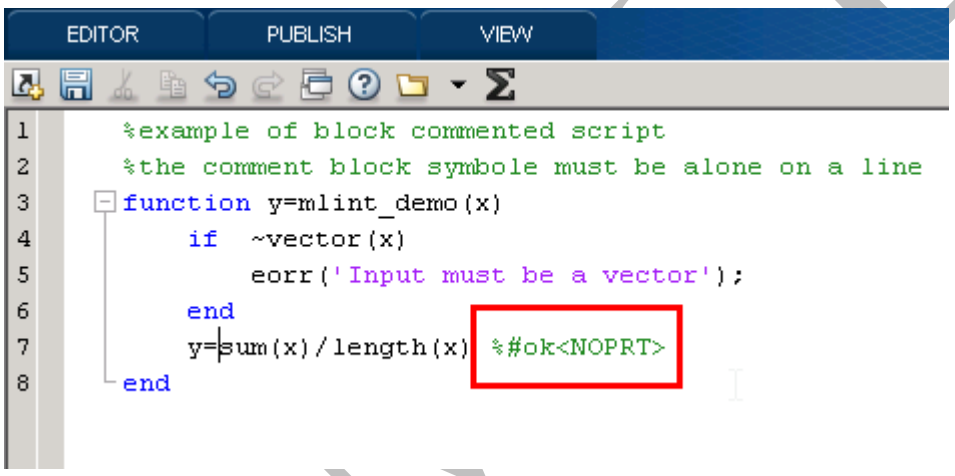


but as you can see there is still an error in reality...

If we ask the M-lint analyzer to ignore something:

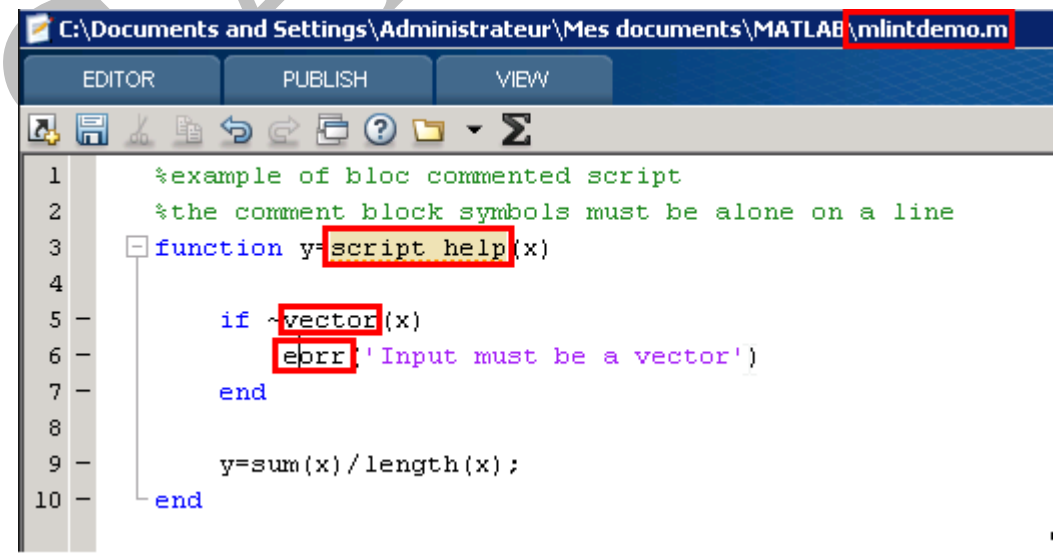


we get:

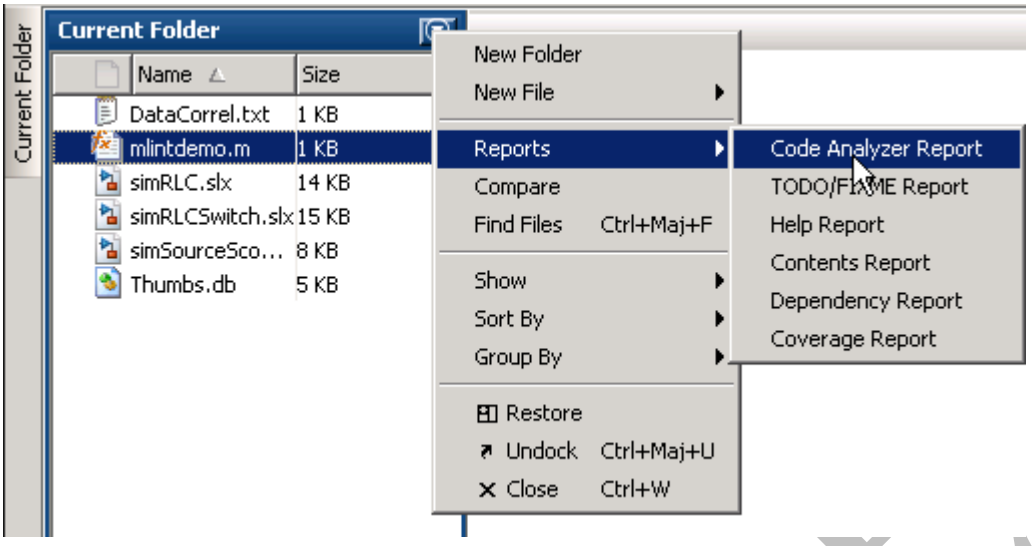


M-lint reports

Consider the following small script with multiple incoherencies and errors:

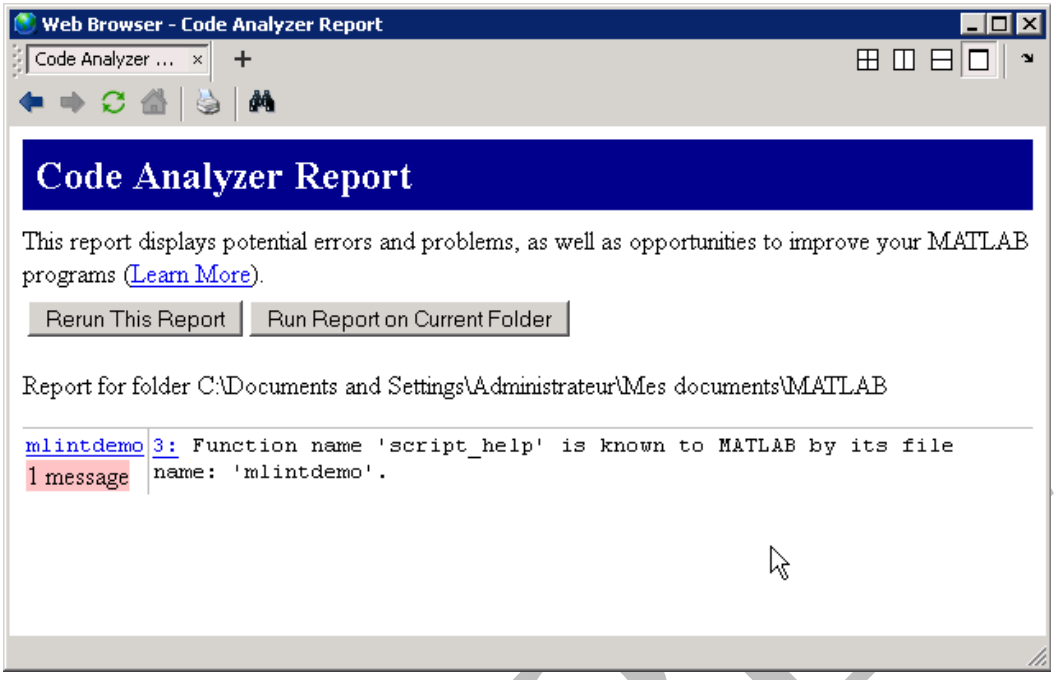


Now take a look the M-Lint **Code Analyzer Report**:

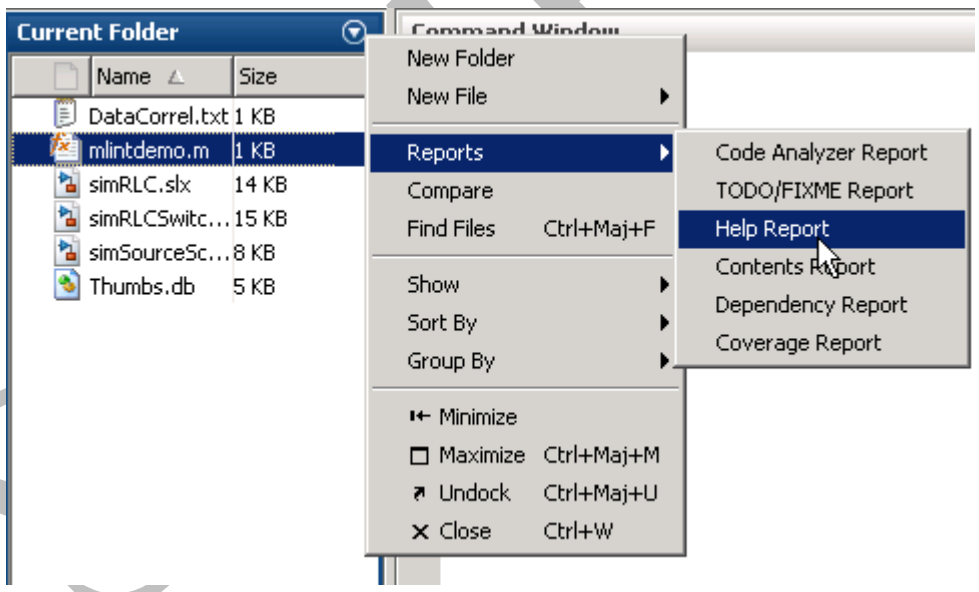

Current Folder
Current Folder
Name
Size
DataCorrel.txt 1 KB
mlintdemo.m 1 KB
simRLC.slx 14 KB
simRLCSwitch.slx 15 KB
simSourceSco... 8 KB
Thumbs.db 5 KB
New Folder
New File
Reports
Compare
Find Files Ctrl+Maj+F
Show
Sort By
Group By
Restore
Undock Ctrl+Maj+U
Close Ctrl+W
Code Analyzer Report
TODO/FIXME Report
Help Report
Contents Report
Dependency Report
Coverage Report

SAMPLE

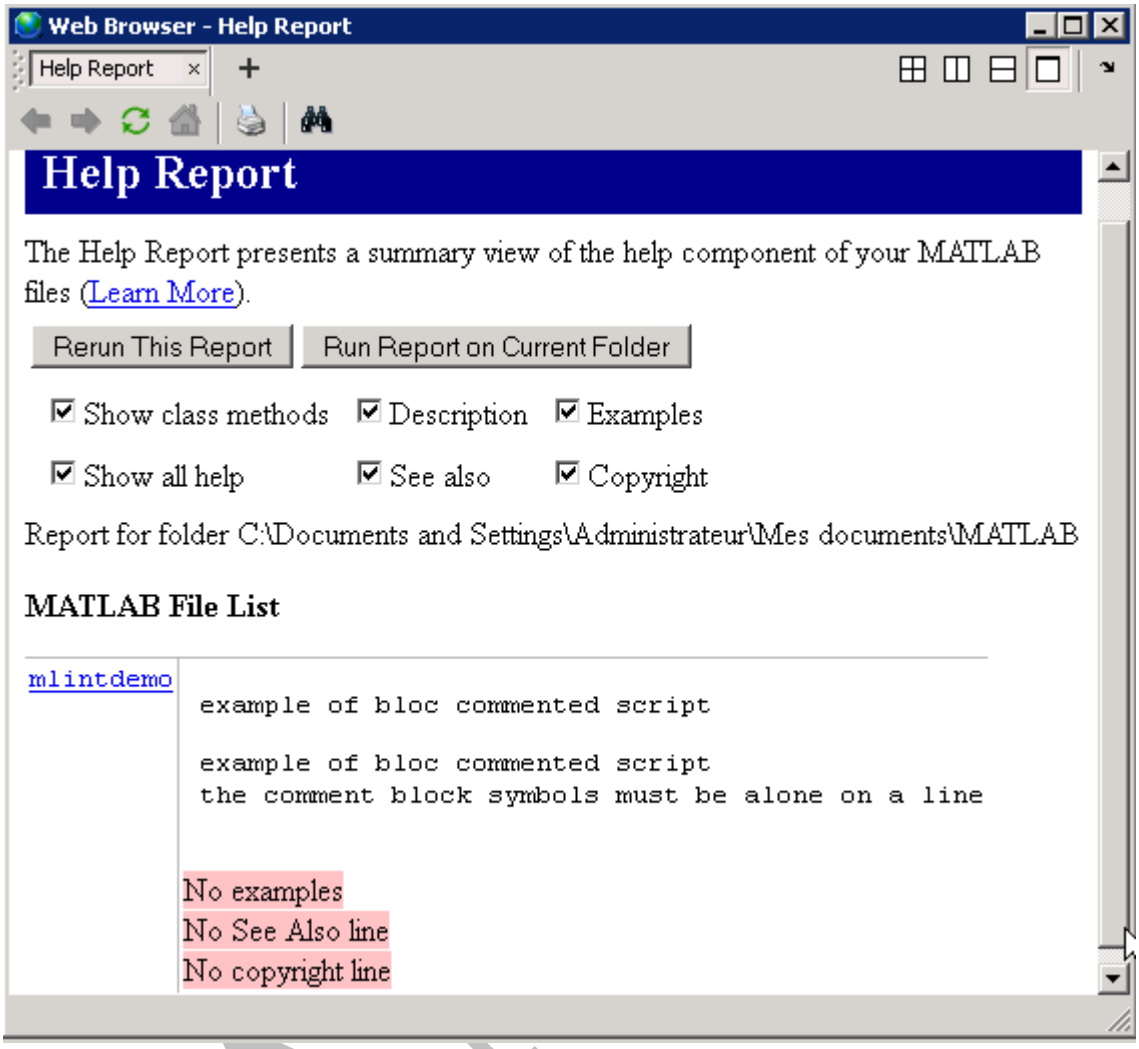
This will give a poor result with a lack in error detection:



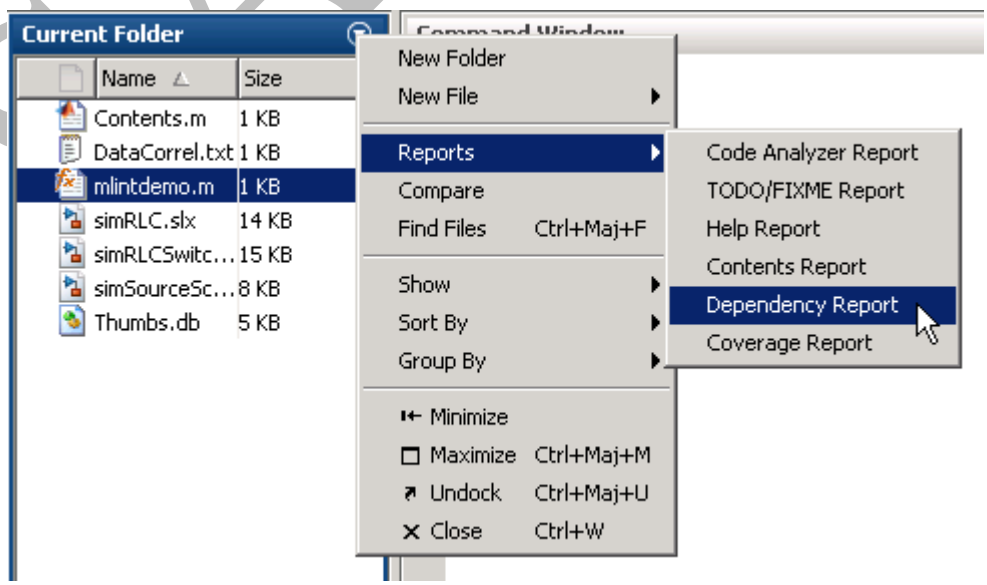
Now consider the **Help Report**:



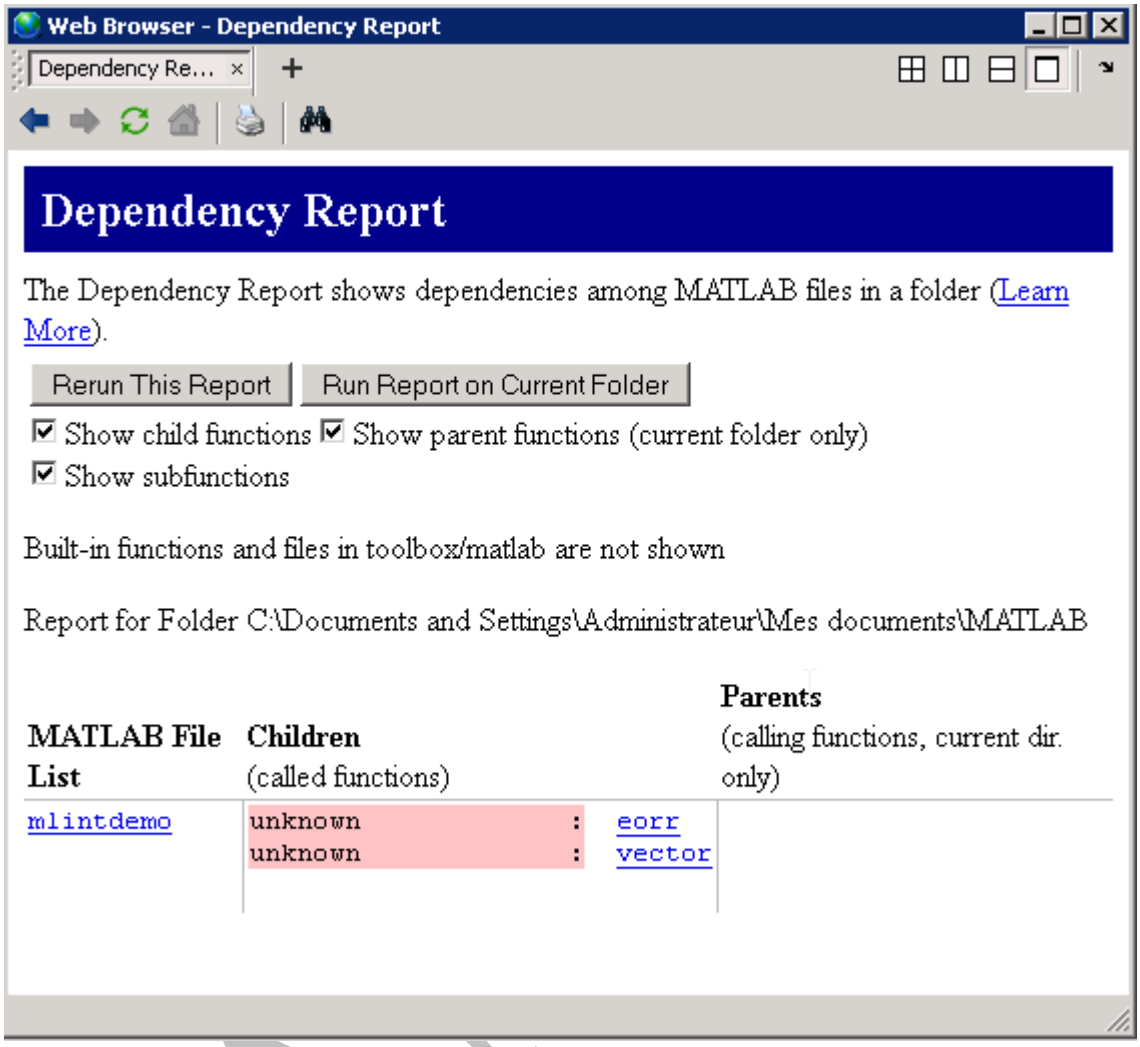
When your script is not correctly documented you get:



Now have a look the **Dependancy Report**:



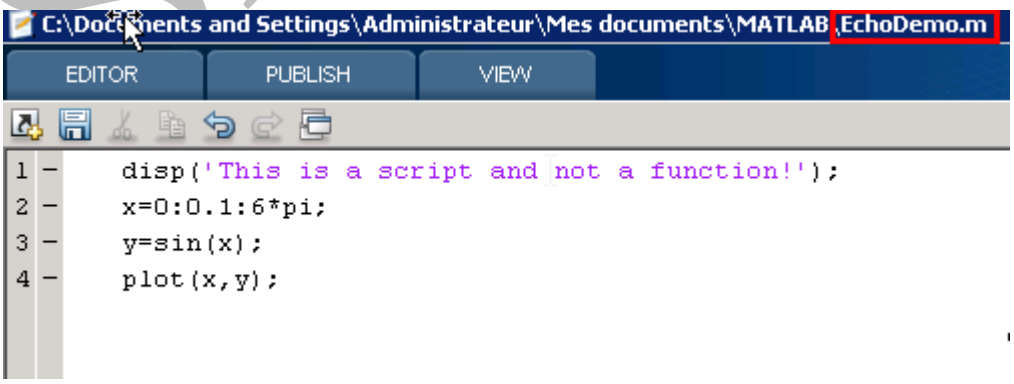
And know we get the error indication about our two misspelled functions:



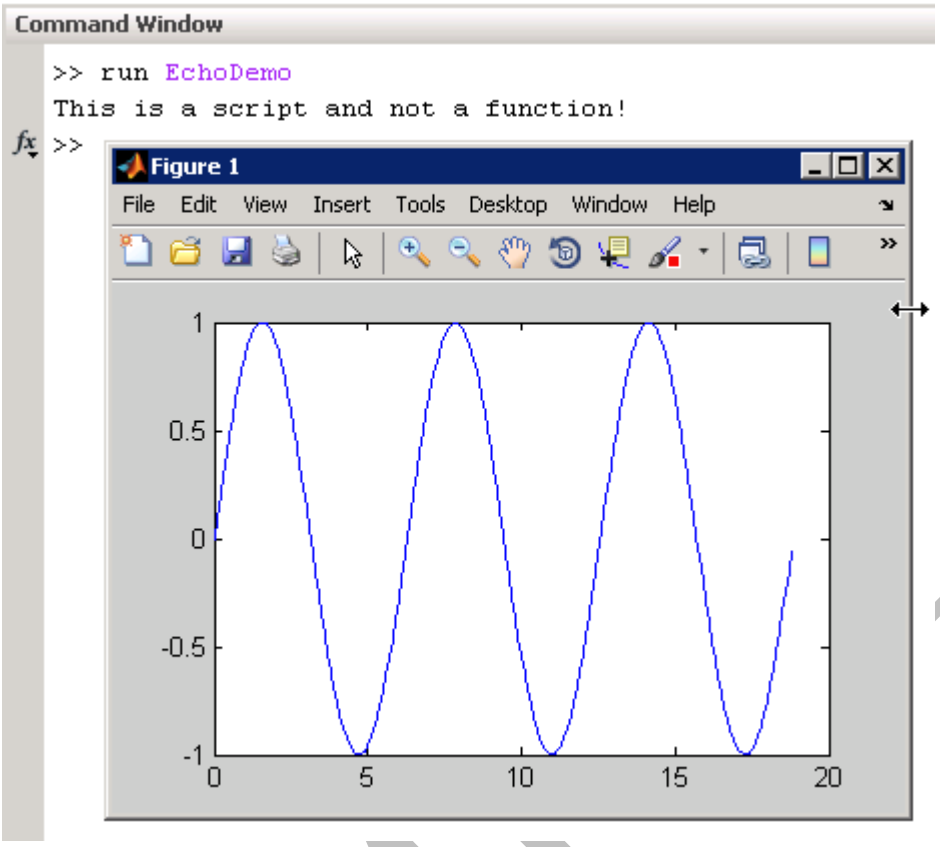
Echo on/Echo off

For simple scripts

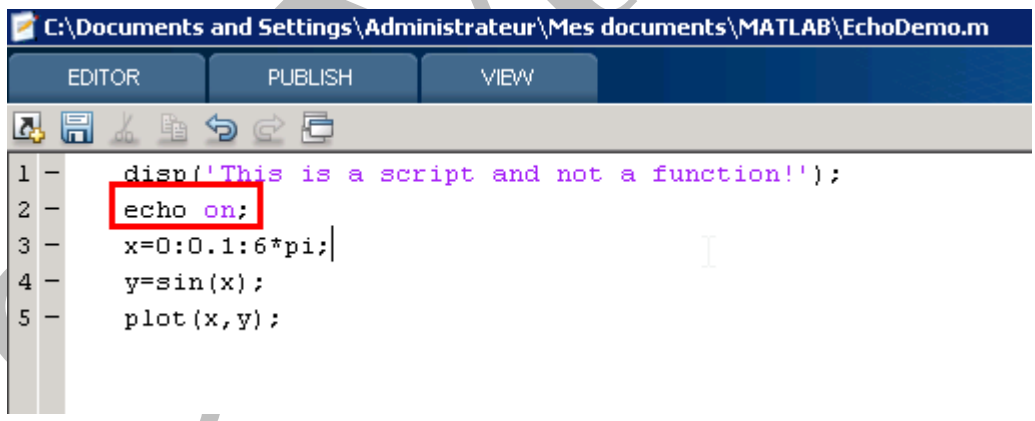
Suppose we have the following script:



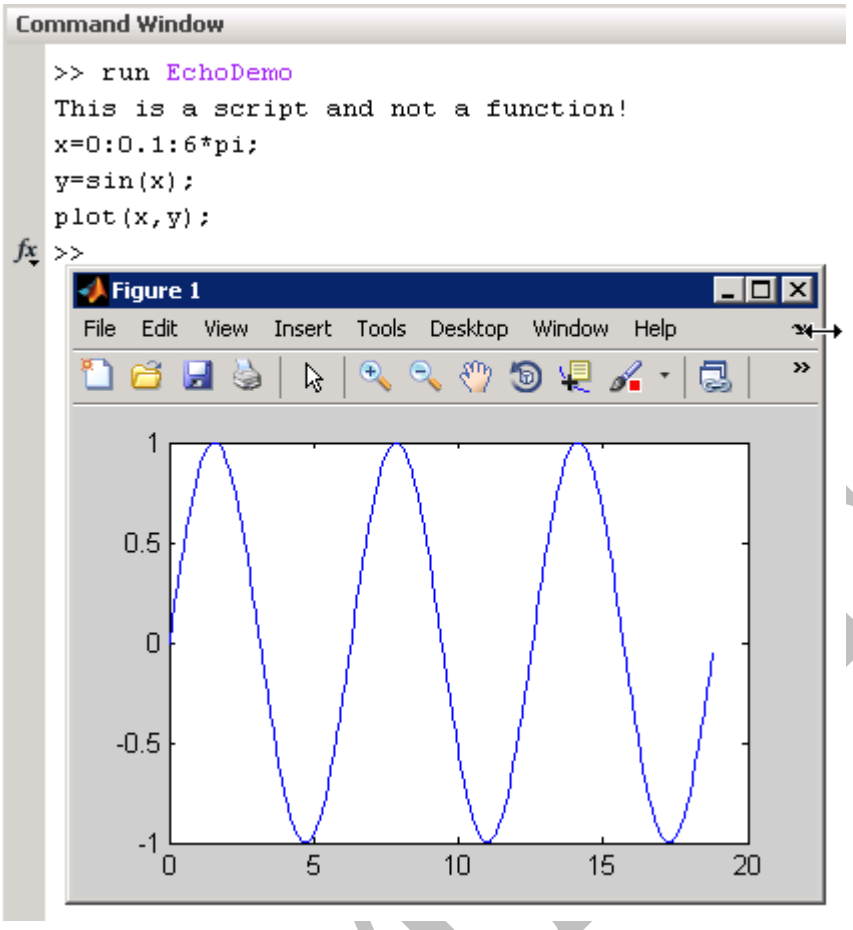
We run it:



We want to see the inside of the script when we run it without having to open/edit the script file. For this purpose we can use the **echo on** function:

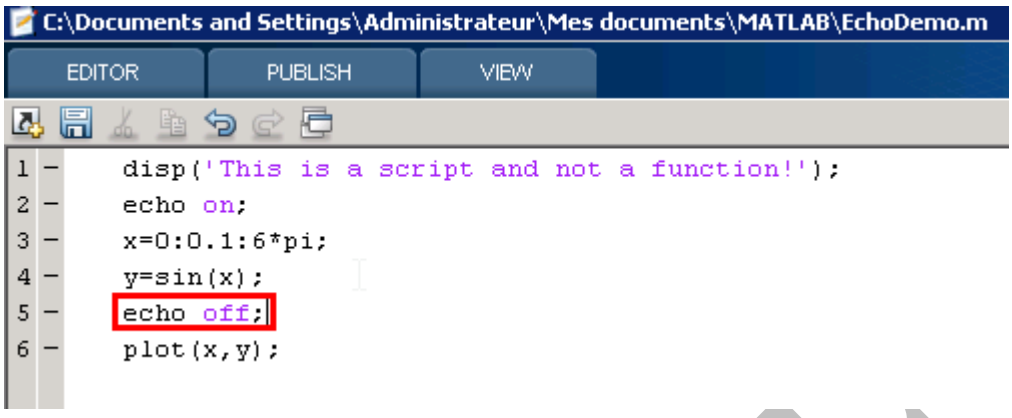


We run it and as you can see it, all the details are printed out:

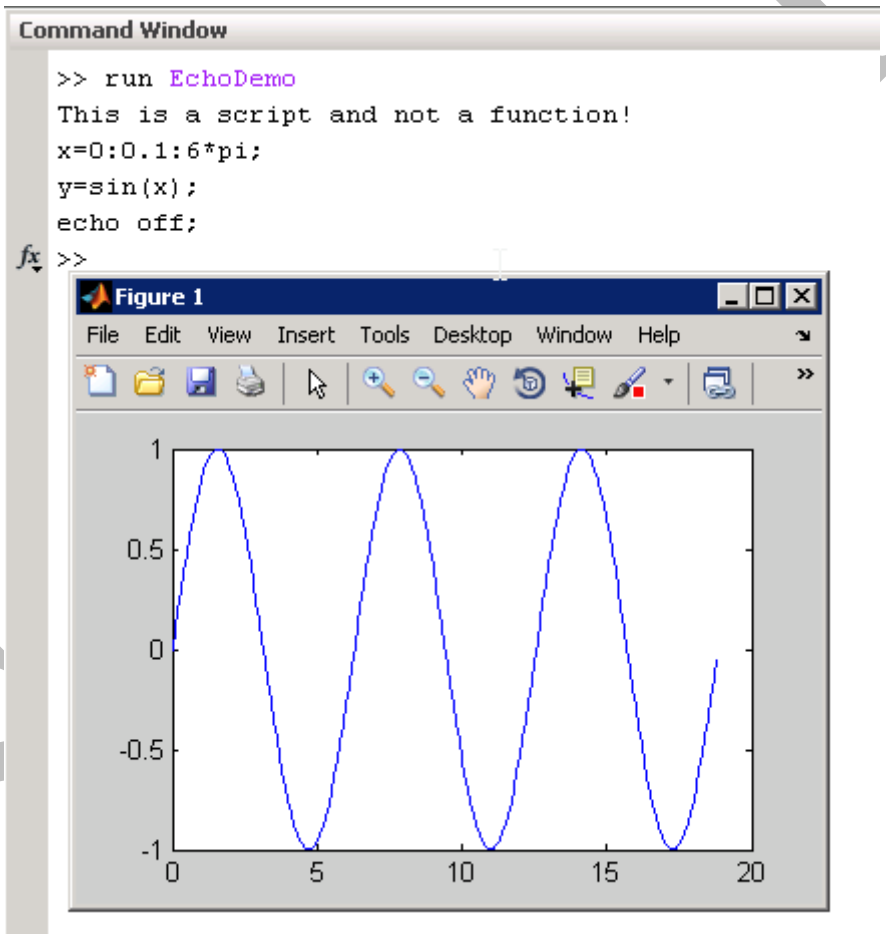

Command Window
>> run EchoDemo
This is a script and not a function!
x=0:0.1:6*pi;
y=sin(x);
plot(x,y);
>>
Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.5
0
-0.5
-1
0
5
10
15
20

SAMPLE

Or if we cancel the **echo on** in the script using the **echo off** command:



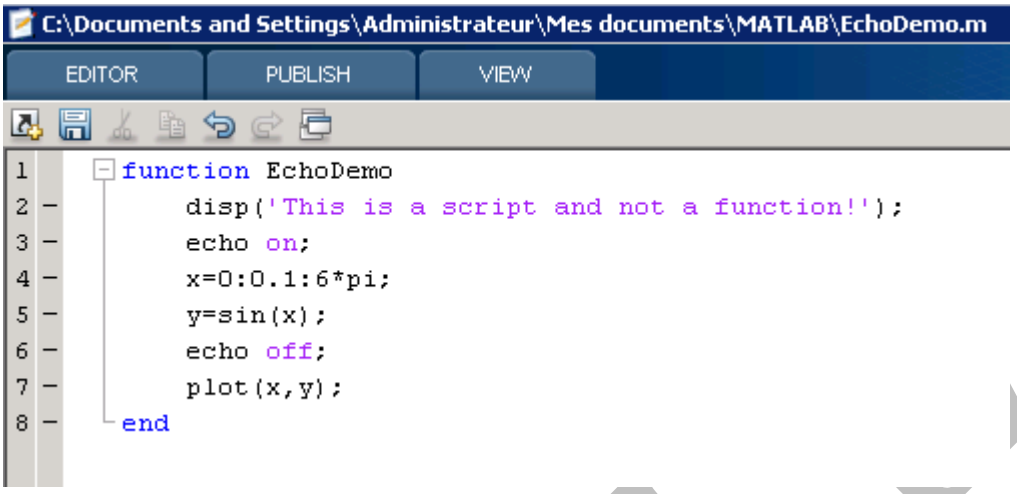
Then we run it again:

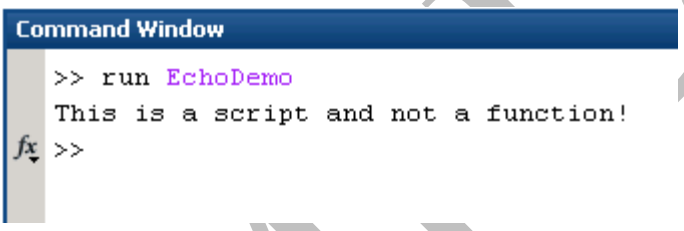


Works well!

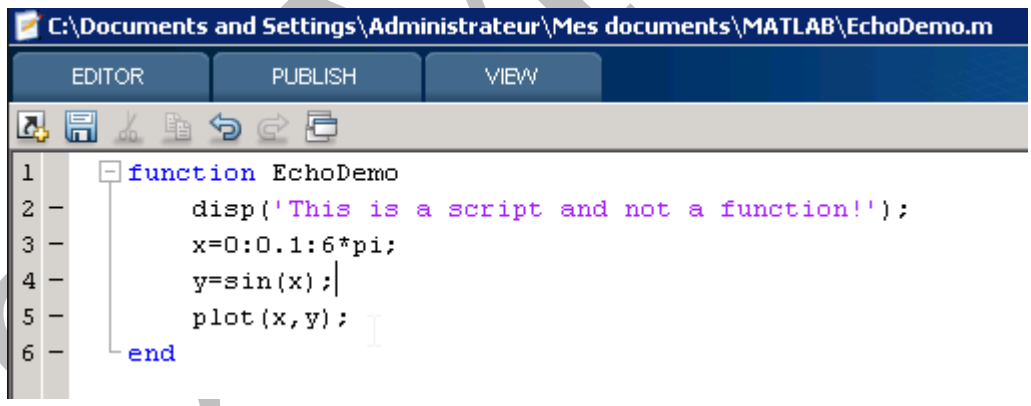
For functions scripts

If we change the previous script into a function wee see that the **echo on** has no effect:

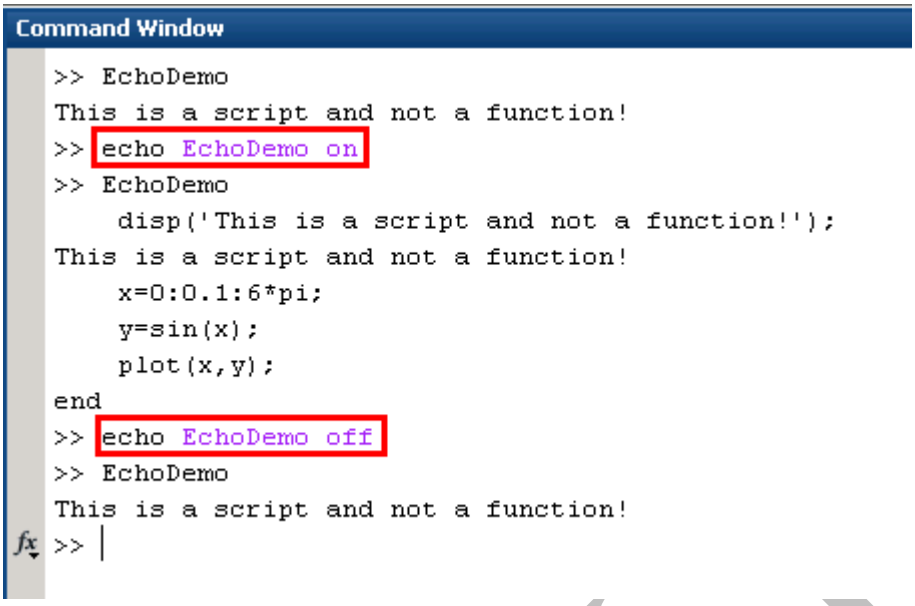




Then we can remove the **echo on** and **echo off** because there are useless:



To activate the echo on a function we have to use the command **echo *nameOfFunction* on** from the *Command Window*:



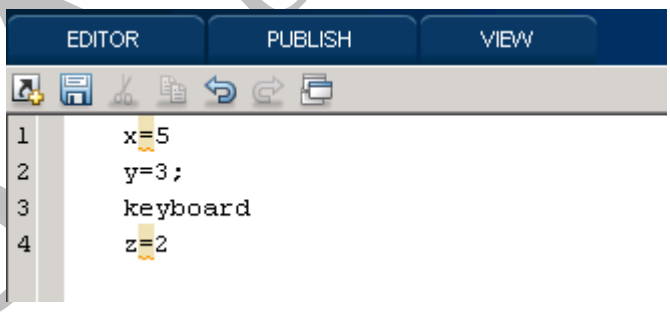
And to stop to echo this function each time we run it, we just write **echo *nameOfFunction* off.**

Manipulations during a run

When you run a script you can't type commands while the script is running. If you write a **pause(...)** statement in the script this won't help because you won't be able to execute immediately what you write in the *Command Window*.

The only way to stop a code and the same time to write something in *Command Window* is to use de **keyboard** command.

For example if you execute this script:

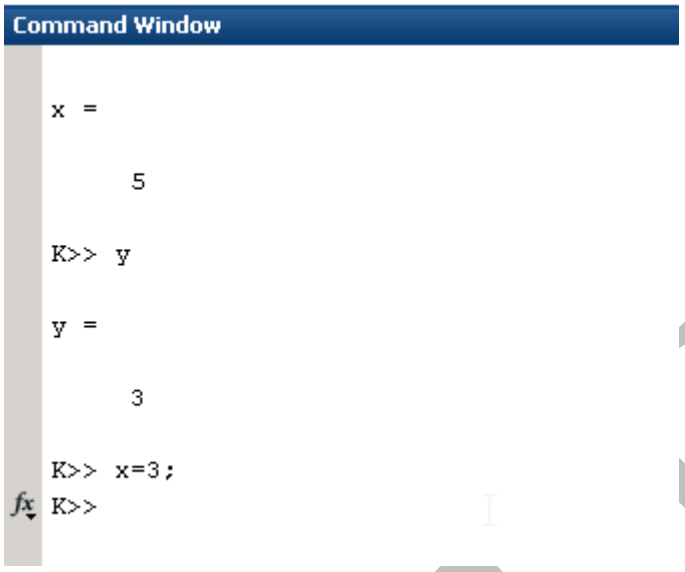


You will get:

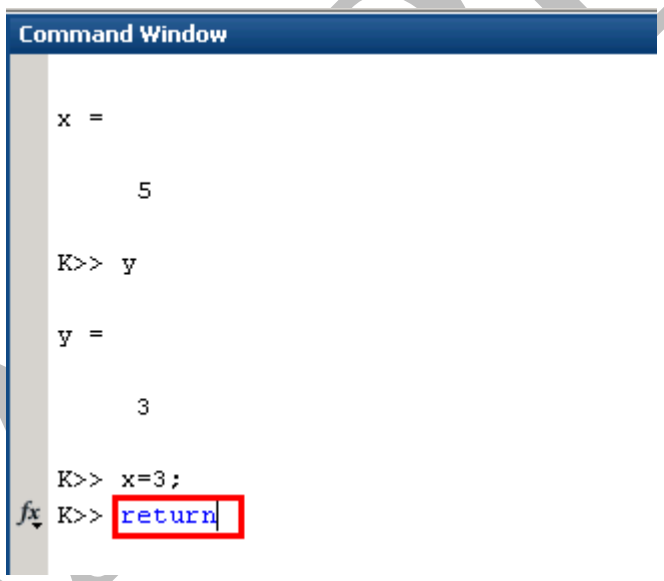


SAMPLE

Then you can type what you want and execute it immediately:



To continue to run the script, you just type **return**:



And that's it.

Debugging (db...) functions

Consider the following script:

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\NPM.m

EDITOR PUBLISH VIEW

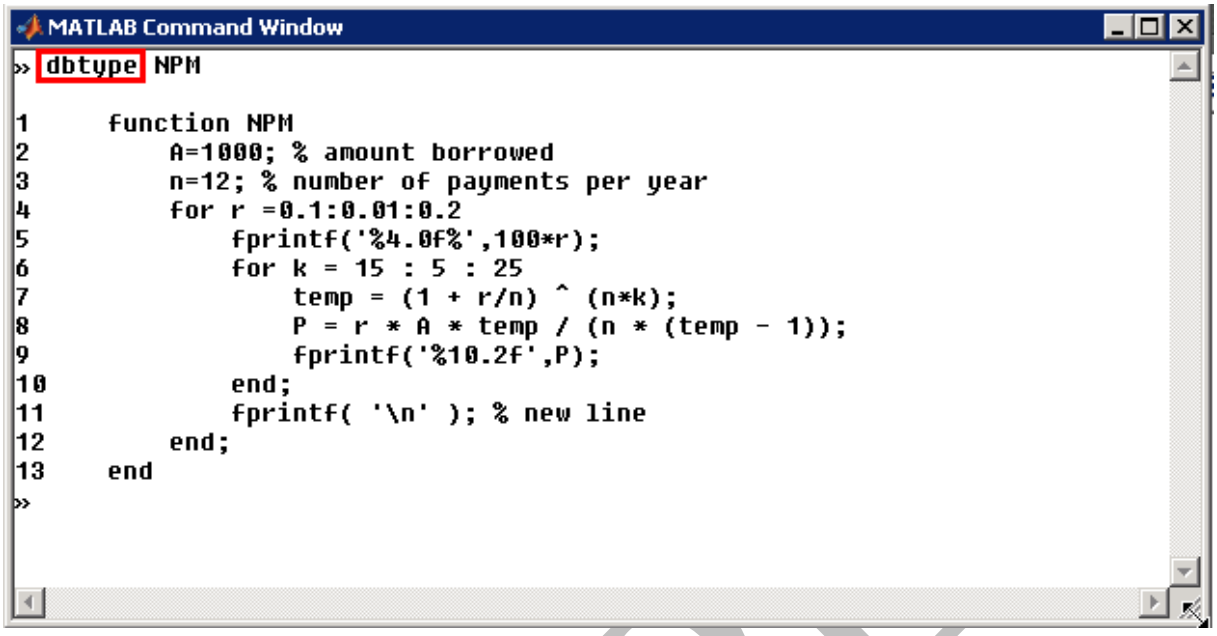
```
1   function NPM
2       A=1000; % amount borrowed
3       n=12; % number of payments per year
4       for r =0.1:0.01:0.2
5           fprintf('%4.0f%',100*r);
6           for k = 15 : 5 : 25
7               temp = (1 + r/n) ^ (n*k);
8               P = r * A * temp / (n * (temp - 1));
9               fprintf('%10.2f',P);
10          end;
11          fprintf( '\n' ); % new line
12      end;
13  end
```

If you run it you will simply get:

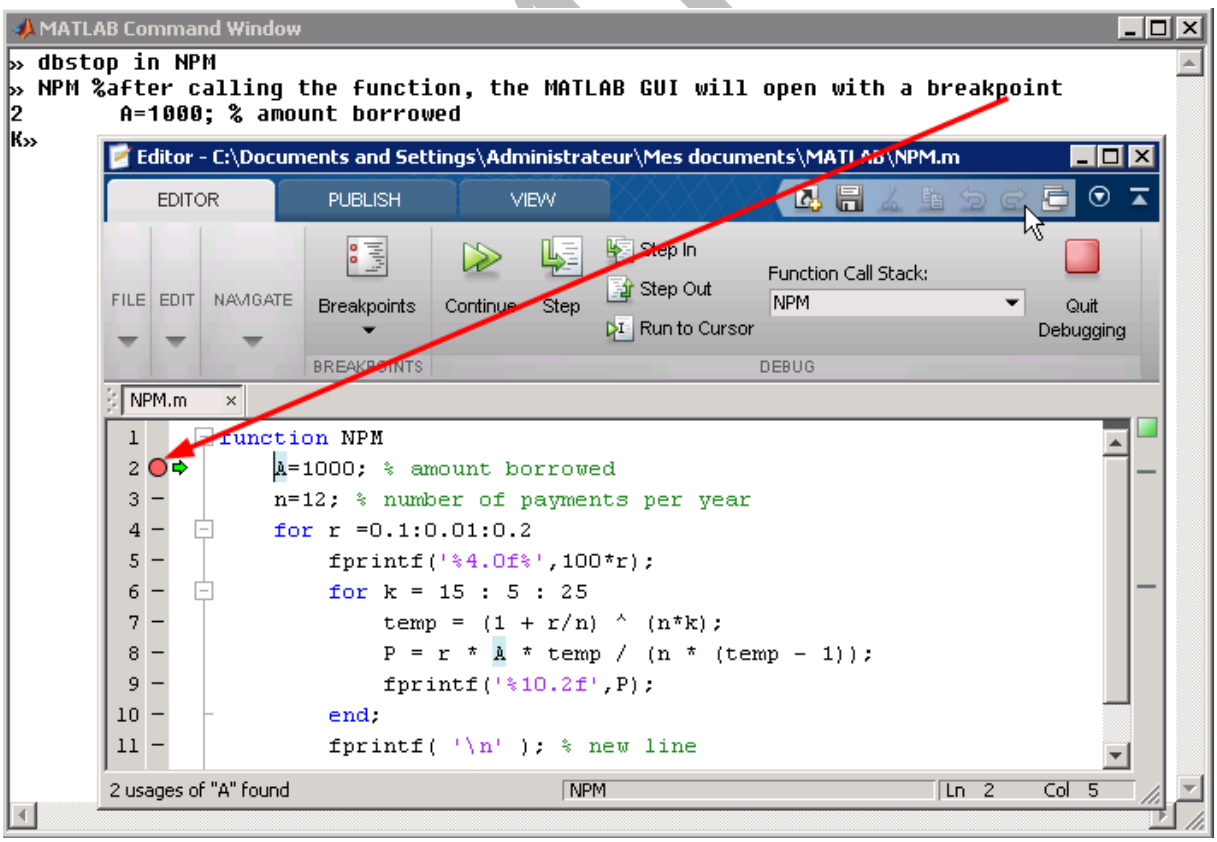
MATLAB Command Window

```
» NPM
  10      10.75       9.65       9.09
  11      11.37      10.32       9.80
  12      12.00      11.01      10.53
  13      12.65      11.72      11.28
  14      13.32      12.44      12.04
  15      14.00      13.17      12.81
  16      14.69      13.91      13.59
  17      15.39      14.67      14.38
  18      16.10      15.43      15.17
  19      16.83      16.21      15.98
  20      17.56      16.99      16.78
»
```

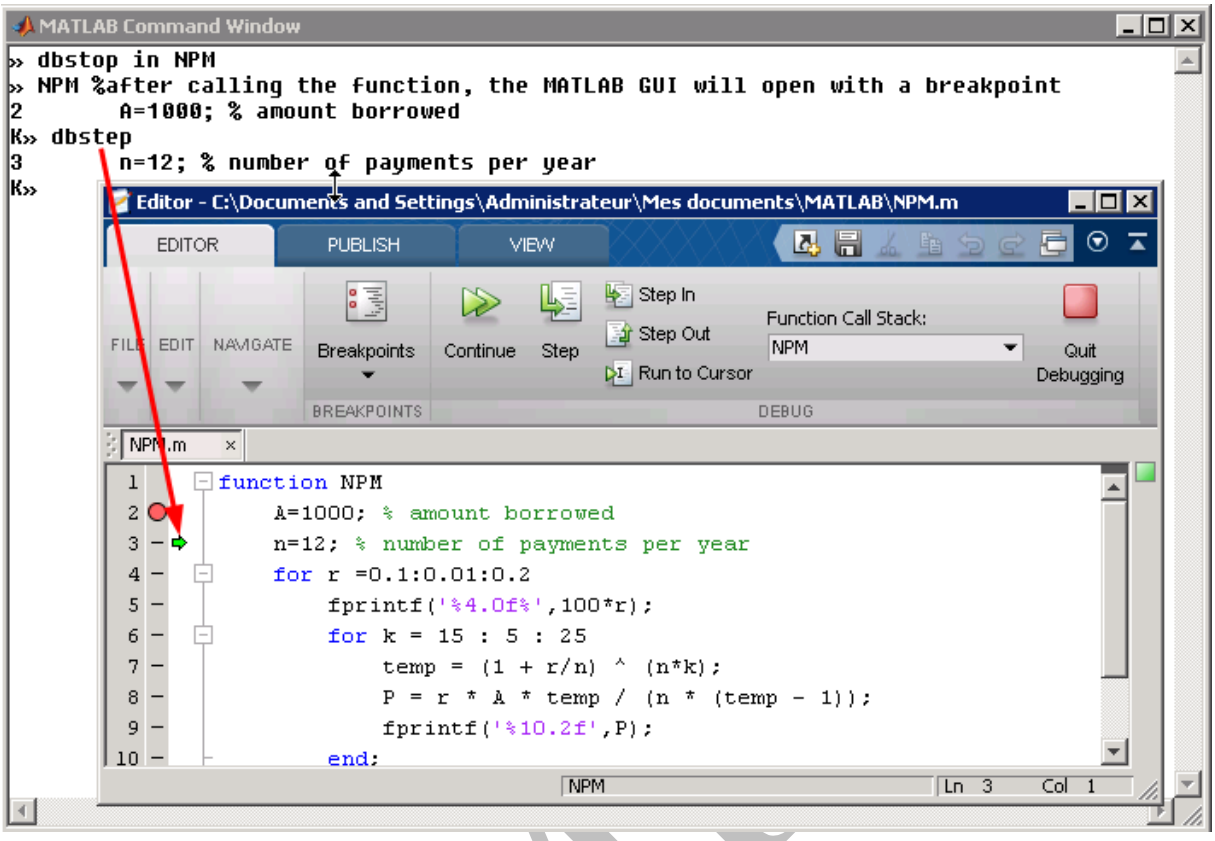
Now if we use the command **dbtype** we get a result that is similar to **echo .... on** with the difference that we see the line number and also the tabulation structure of the original script:



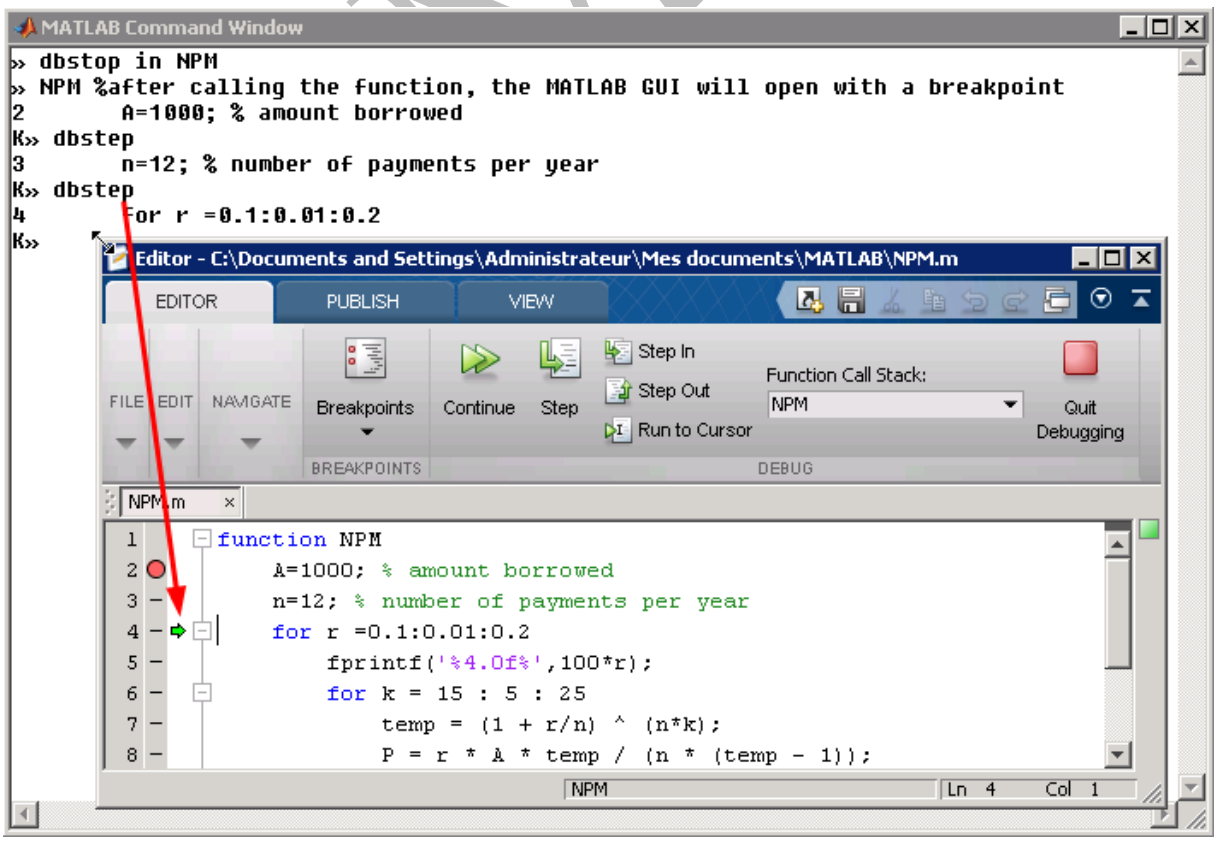
If we use the command **dbstop** the function will be executed line by line with the keyboard mode activated and the scripts editor automatically opens:



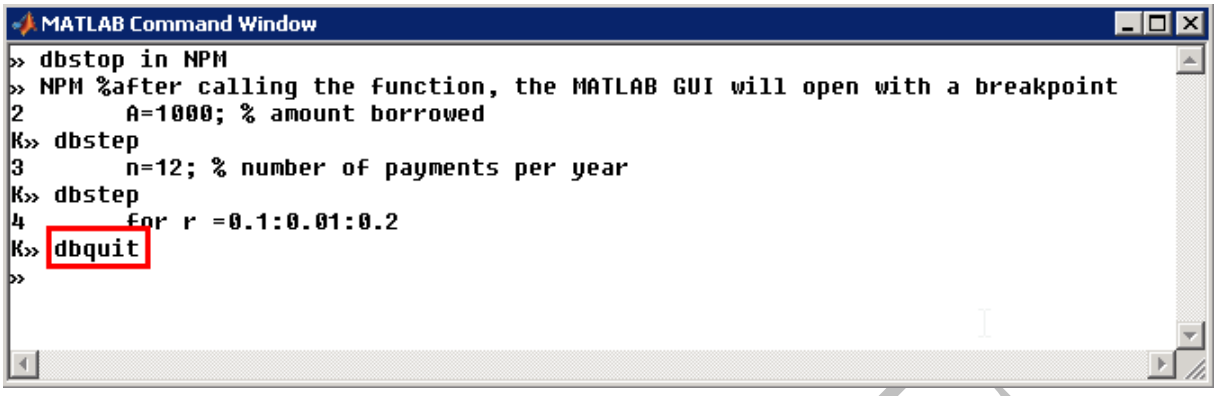
To go further we write **dbstep** and the next line of the function will be executed and show:



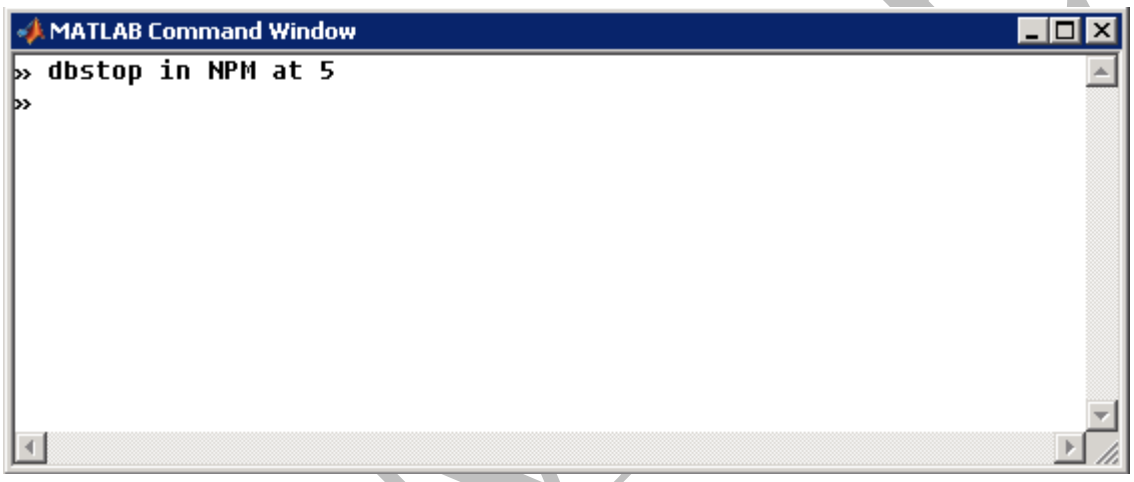
and so on...:



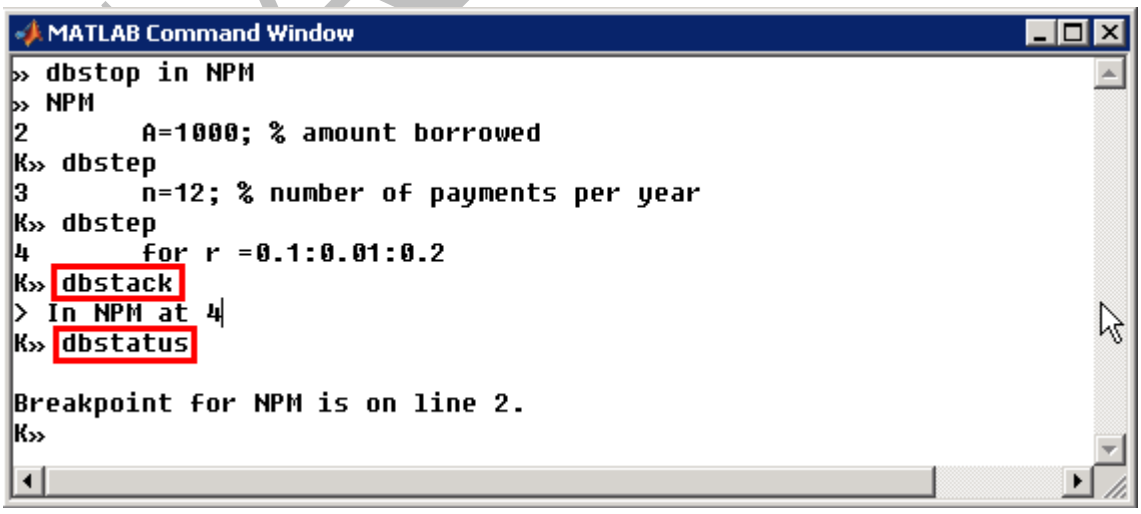
And if we want to stop the debugging and the script we use the command **dbquit**:



You can also put a break point at a given desired line:

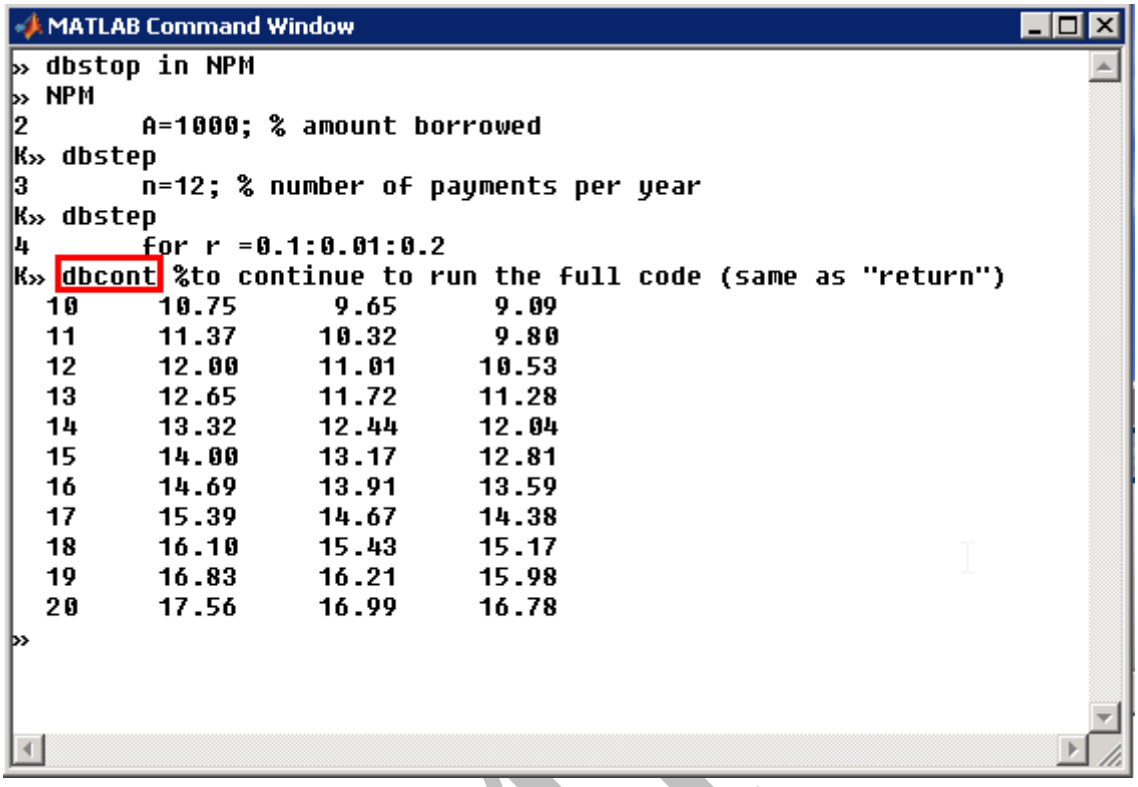


The command **dbstack** gives you the possibility to know where you are in the debugging of your script by returning the line number and **dbstatus** returns you the line where the first breakpoint has started the debugging:



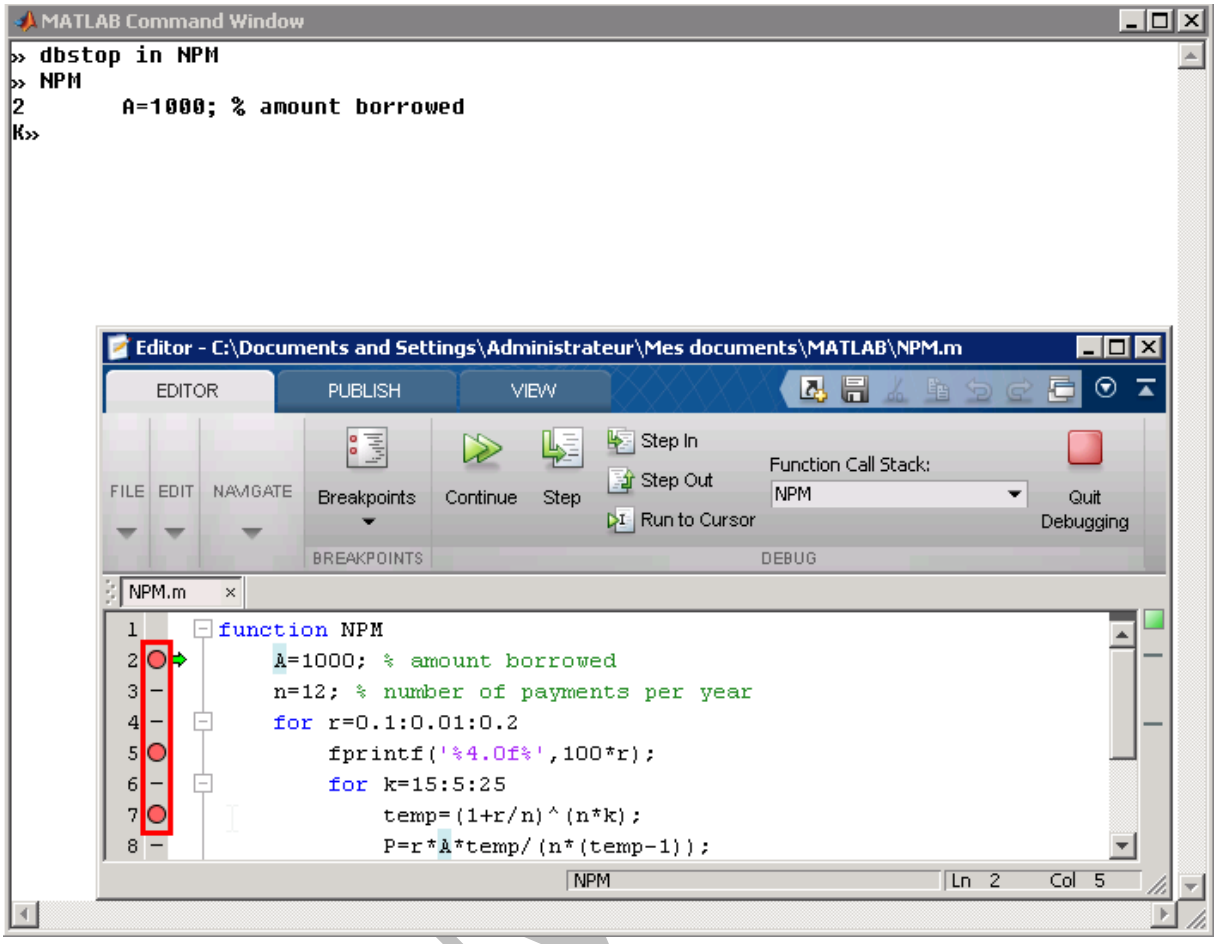
This two last function are not used a lot.

Now when you debug a script you want sometimes to run the script until the end without stopping it again. Then you just have to use the command **dbcont**:



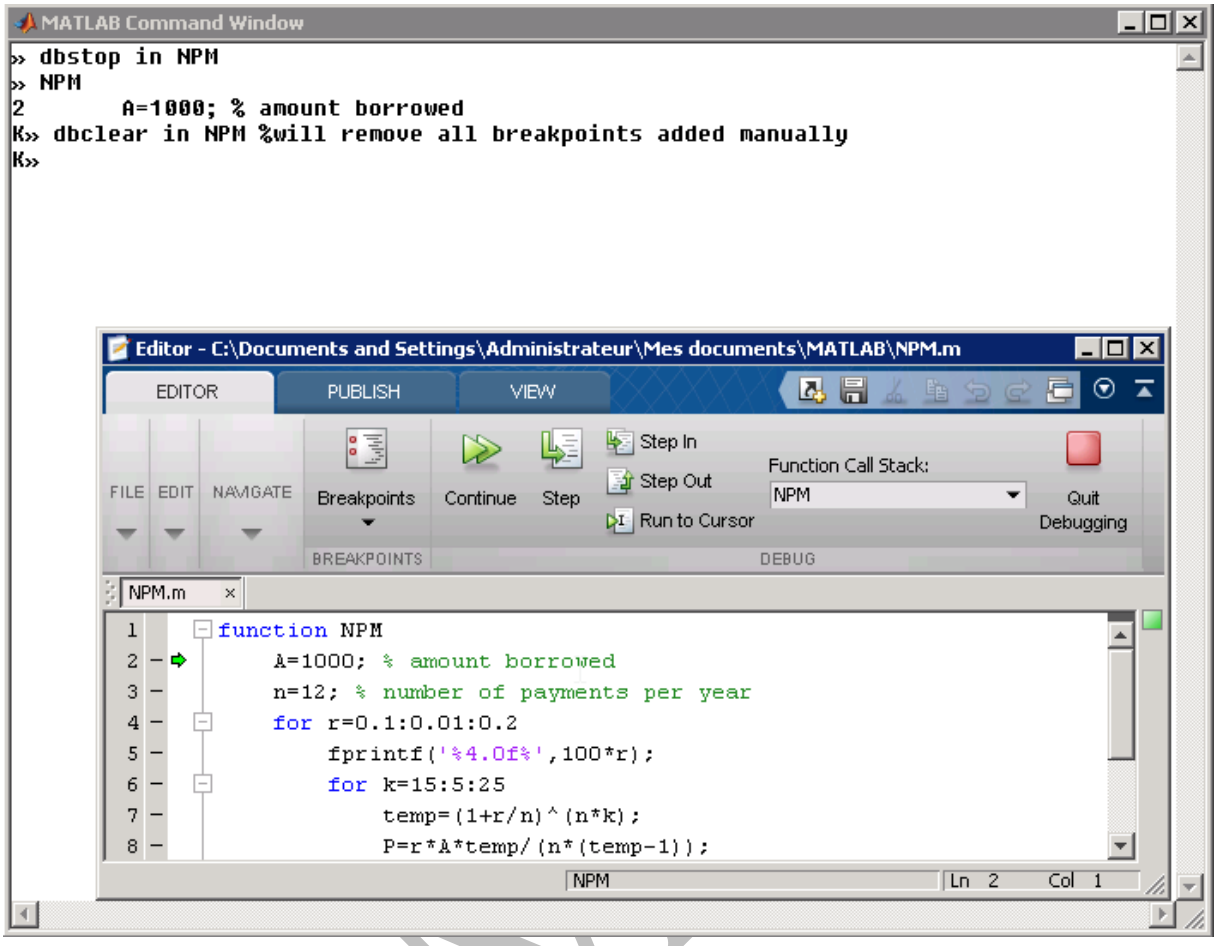
SAMPLE

Now have a closer look to the breakpoint that **dbstop** add automatically on line number 2 of the script and let us add other breakpoints manually as you can see here below:



SAM

If you use the command **dbclear**, all breakpoints will be deleted:



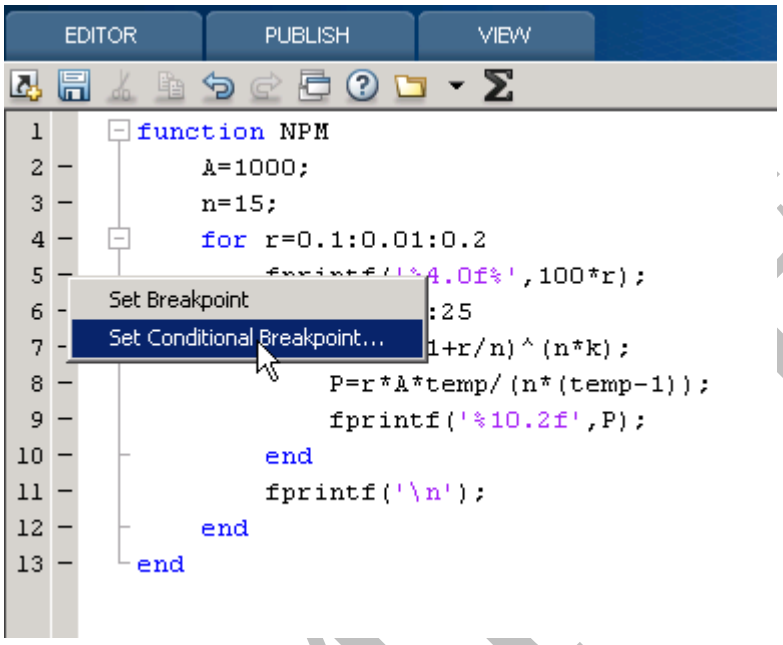
You can use the **s=dbstatus** syntax and then save **s** to save the current breakpoints to a MAT-file. At a later time, you can load s and restore the breakpoints using **dbstop(s)**.

SAM

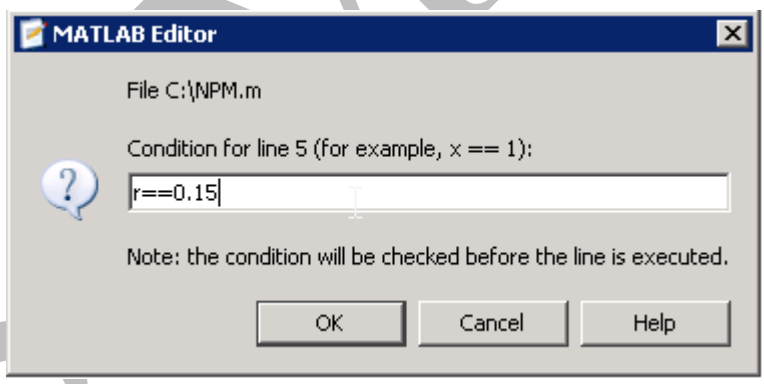
Add conditional breakpoints

This is in my point of view the most important debugging manipulation. To define such a breakpoint proceed as following:

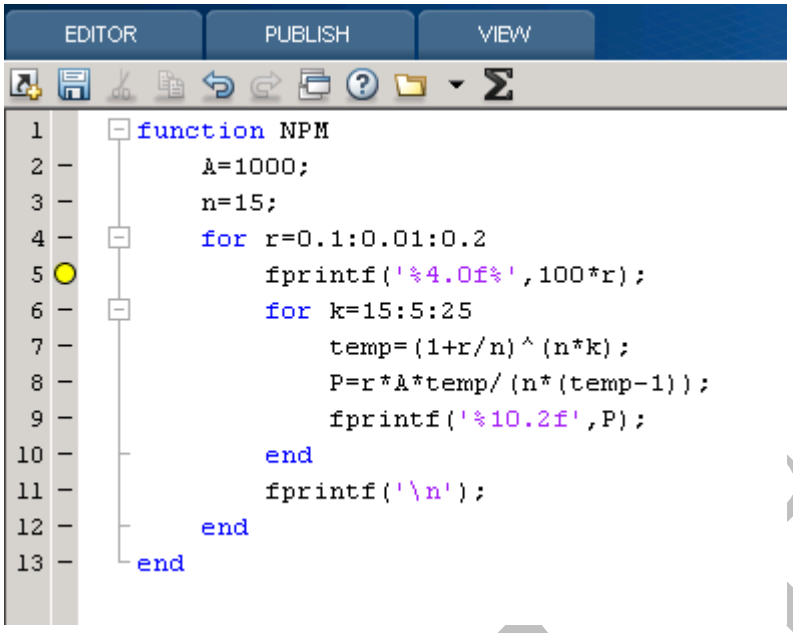
Make a right click at the line level where you want a condition breakpoint:



Then you type what you want:



And when you validate by **OK**, you get a yellow breakpoint:




```
function NPM
    A=1000;
    n=15;
    for r=0.1:0.01:0.2
        fprintf('%4.0f%',100*r);
        for k=15:5:25
            temp=(1+r/n)^(n*k);
            P=r*A*temp/(n*(temp-1));
            fprintf('%10.2f',P);
        end
        fprintf('\n');
    end
end
```

SAMPLE

Create a more complex wait bar

A lot of MATLAB users and some developers... absolutely wants a progress bar even if it slow down dramatically the running time of the a script as we already saw it (see page 209).

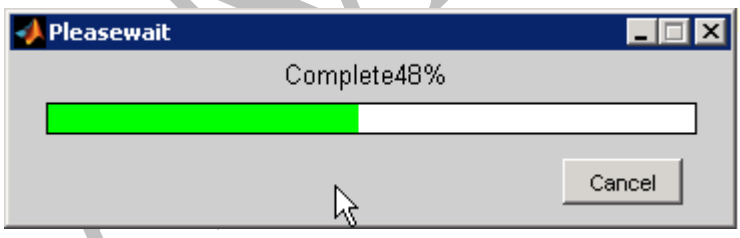
The problem that we saw the first time with the **waitbar( )** function is that it is very limited because we could not change the color of the progress bar and not add a button to cancel the whole running script.

Now that we are studying scripts it is time to introduce a more complex **waitbar( )** and as you can see below, the syntax is not very intuitive:

```
EDITOR  PUBLISH  VIEW

1     steps = 100;
2     wb = waitbar(0,'Initializing...','Name','Pleasewait',...
3         'createCancelBtn','setappdata(gcbf,''canceling'',1)');
4     set(findobj(wb,'Type','Patch'),'EdgeColor','green','FaceColor','green')
5     setappdata(wb,'canceling',0)
6     for step = 1:steps;
7         if getappdata(wb,'canceling'); break; end
8         waitbar(step/steps,wb,['Complete' num2str(step*100/steps) '%']);
9        % computation
10    end
11    delete(wb);
```

This will result in:



Import an XML file

Importing XML feed is something always more and more frequent in computing developments. Because the MATLAB import wizard is not able at this day to import XML files we have waited until now to introduce the script to do such a job.

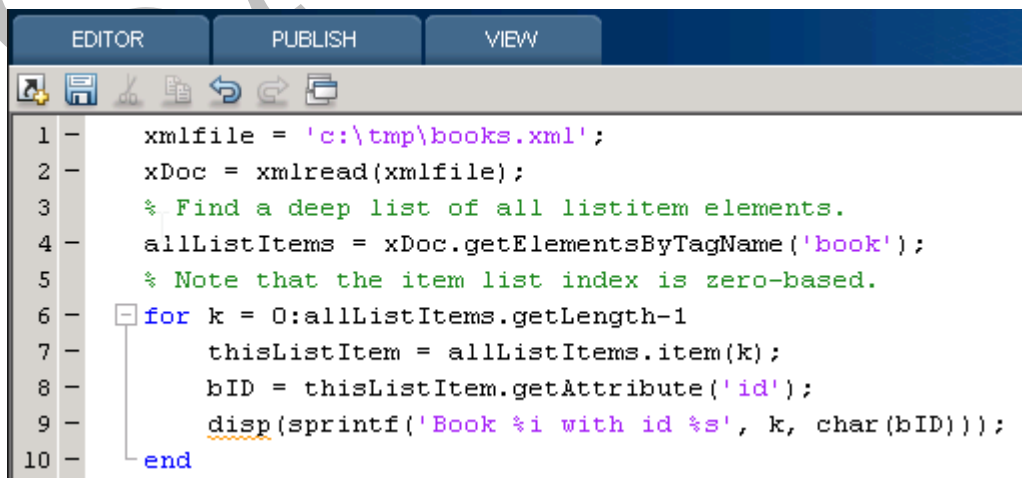
For this purpose consider the following typical XML file:




```
<?xml version="1.0" ?>
- <catalog>
  - <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications with XML.</description>
    </book>
  - <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.</description>
    </book>
  - <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.</description>
    </book>
  - <book id="bk104">
      <author>Corets, Eva</author>
      <title>Oberon's Legacy</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2001-03-10</publish_date>
```

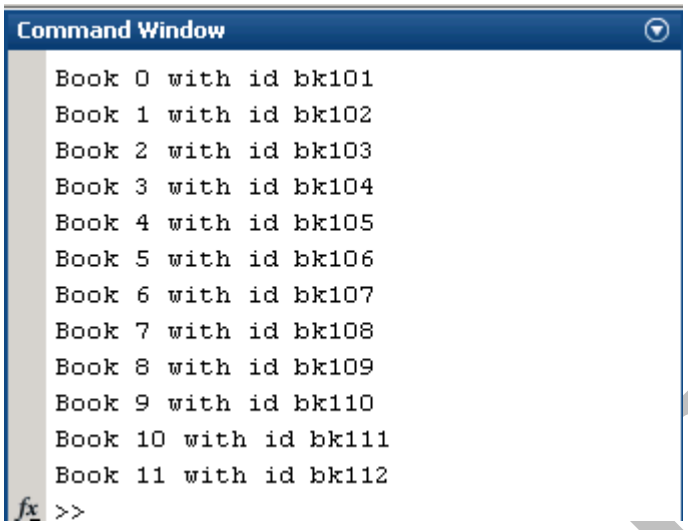
And the following script:

xmlfile='c:\tmp

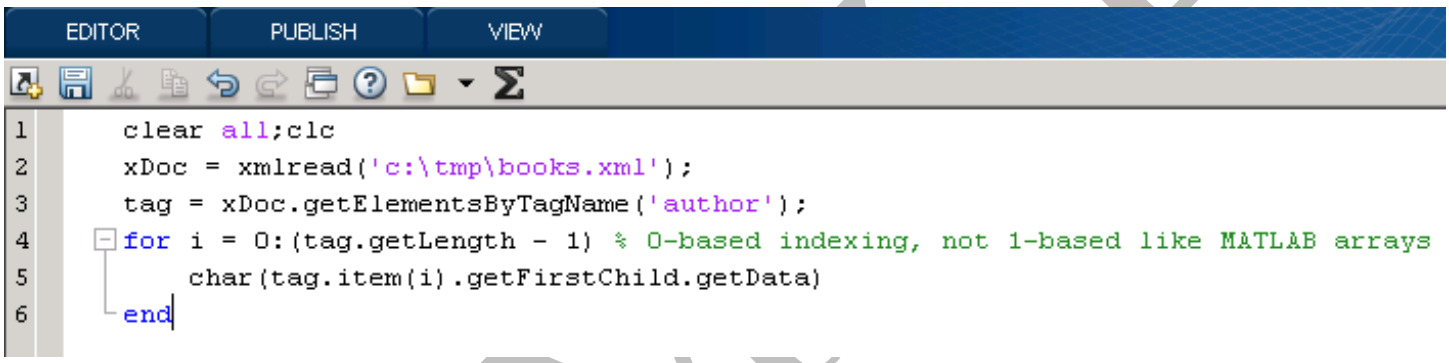



```
xmlfile = 'c:\tmp\books.xml';
xDoc = xmlread(xmlfile);
% Find a deep list of all listitem elements.
allListItems = xDoc.getElementsByTagName('book');
% Note that the item list index is zero-based.
for k = 0:allListItems.getLength-1
    thisListItem = allListItems.item(k);
    bID = thisListItem.getAttribute('id');
    disp(sprintf('Book %i with id %s', k, char(bID)));
end
```

The result will be:



And now to read tag values and not only attributes:



It gives as expected:

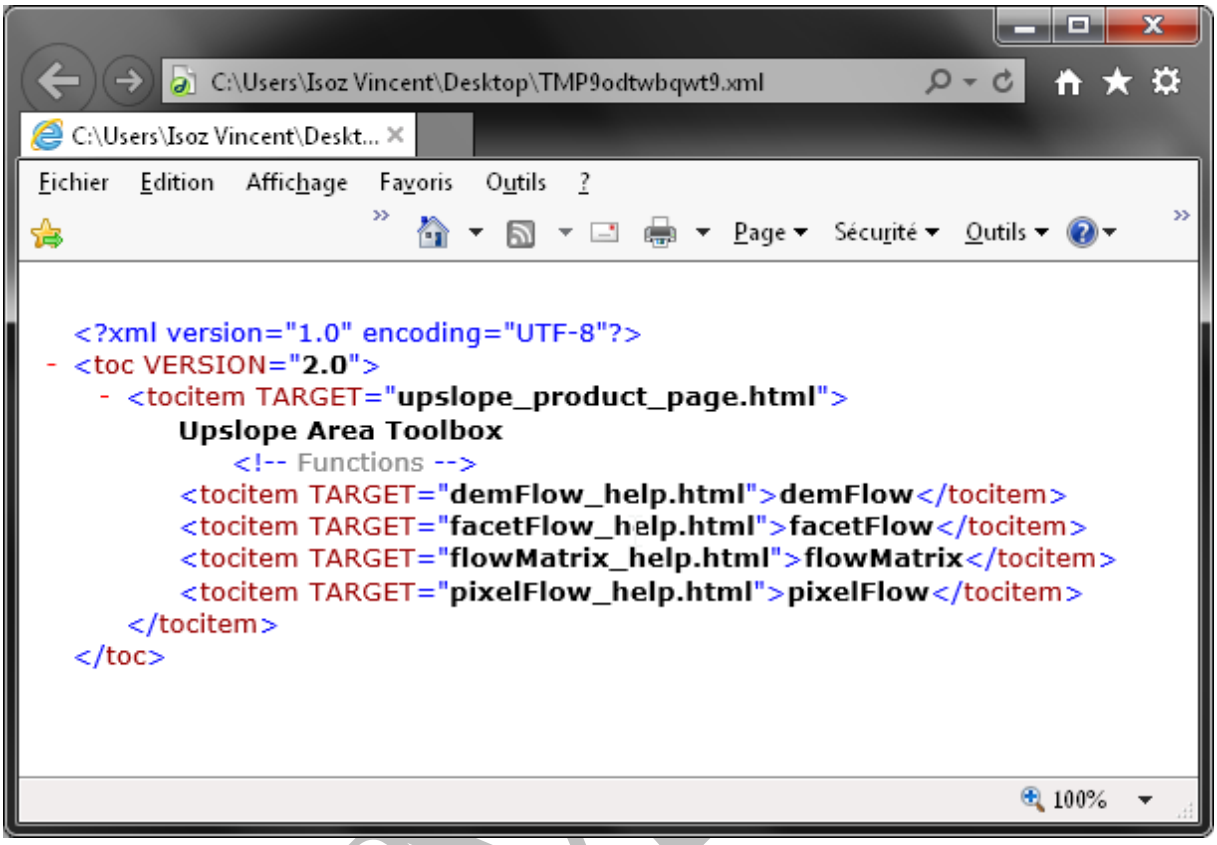
SAMPLE


Command Window
ans =
Gambardella, Matthew
ans =
Ralls, Kim
ans =
Corets, Eva
ans =
Corets, Eva
ans =
Corets, Eva

SAMPLE

Export an XML file

Suppose we want to export some datas to create the following XML file:



The script will be the following using **xmlwrite( )**:

SAMPLE

EDITOR PUBLISH VIEW

```
docNode = com.mathworks.xml.XMLUtils.createDocument('toc');
toc = docNode.getDocumentElement;
toc.setAttribute('version','2.0');

product = docNode.createElement('tocitem');
product.setAttribute('target','upslope_product_page.html');
product.appendChild(docNode.createTextNode('Upslope Area Toolbox'));
toc.appendChild(product)

product.appendChild(docNode.createComment(' Functions '));

functions = {'demFlow','facetFlow','flowMatrix','pixelFlow'};
for idx = 1:numel(functions)
    curr_node = docNode.createElement('tocitem');

    curr_file = [functions{idx} '_help.html'];
    curr_node.setAttribute('target',curr_file);

    % Child text is the function name.
    curr_node.appendChild(docNode.createTextNode(functions{idx}));
    product.appendChild(curr_node);
end

xmlwrite('info.xml',docNode);
type('info.xml');
```

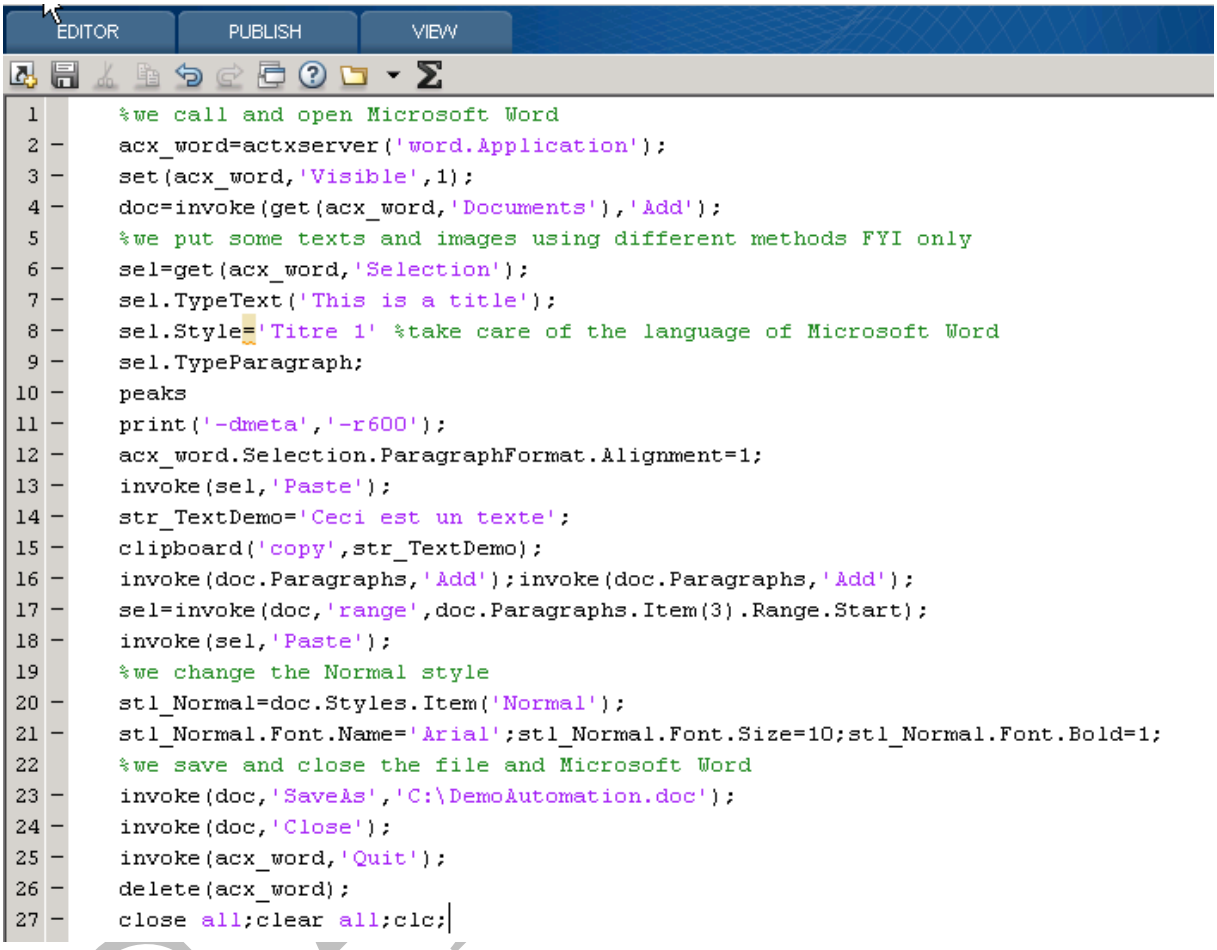
SAM

Automation

Generate a Microsoft Word report

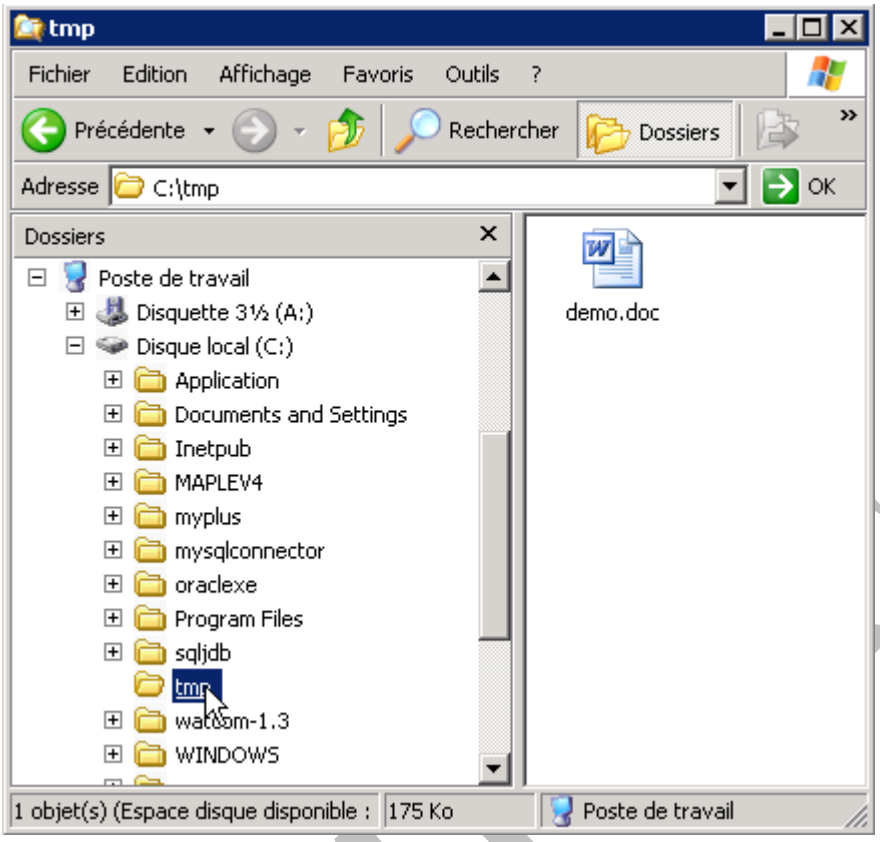
Generating automated Microsoft Word report is something very common.

Here is an example of most common Microsoft Word Automation tasks:

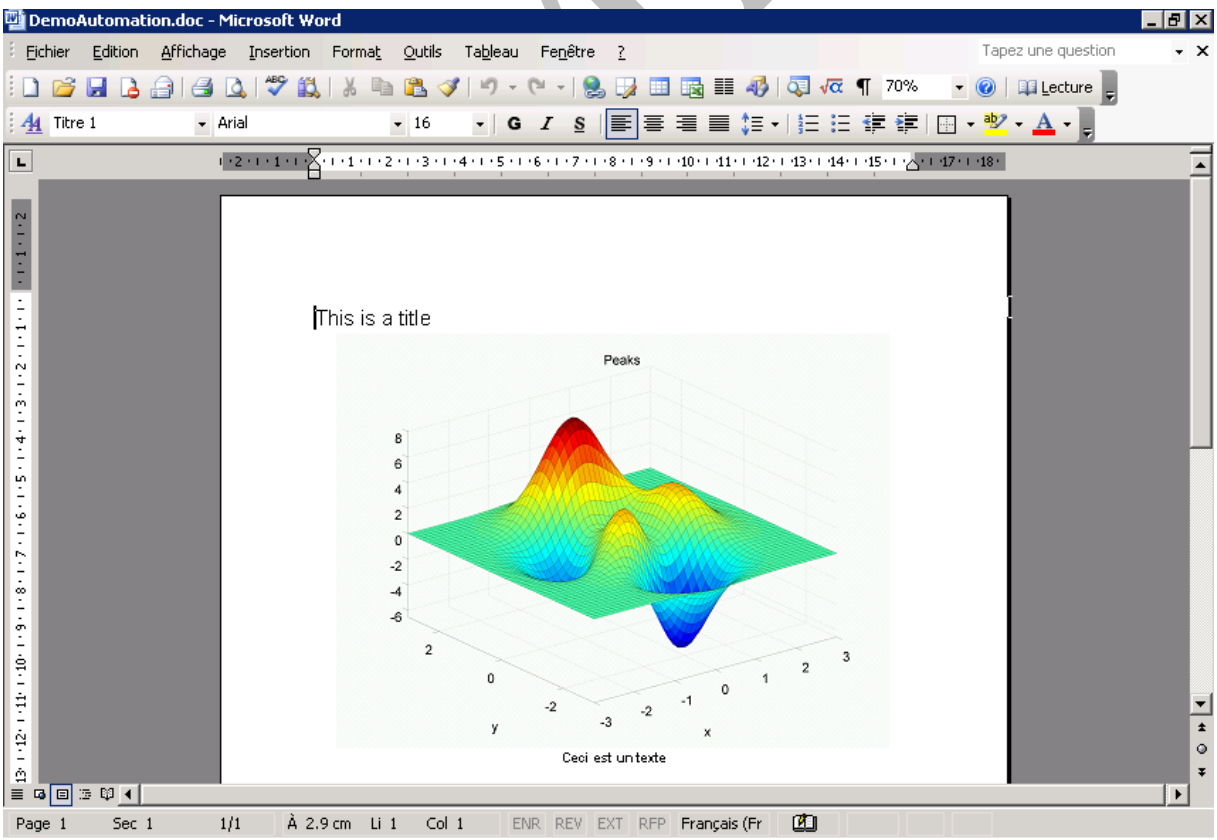



```
%we call and open Microsoft Word
acx_word=actxserver('word.Application');
set(acx_word,'Visible',1);
doc=invoke(get(acx_word,'Documents'),'Add');
%we put some texts and images using different methods FYI only
sel=get(acx_word,'Selection');
sel.TypeText('This is a title');
sel.Style='Titre 1' %take care of the language of Microsoft Word
sel.TypeParagraph;
peaks
print('-dmeta','-r600');
acx_word.Selection.ParagraphFormat.Alignment=1;
invoke(sel,'Paste');
str_TextDemo='Ceci est un texte';
clipboard('copy',str_TextDemo);
invoke(doc.Paragraphs,'Add');invoke(doc.Paragraphs,'Add');
sel=invoke(doc,'range',doc.Paragraphs.Item(3).Range.Start);
invoke(sel,'Paste');
%we change the Normal style
stl_Normal=doc.Styles.Item('Normal');
stl_Normal.Font.Name='Arial';stl_Normal.Font.Size=10;stl_Normal.Font.Bold=1;
%we save and close the file and Microsoft Word
invoke(doc,'SaveAs','C:\DemoAutomation.doc');
invoke(doc,'Close');
invoke(acx_word,'Quit');
delete(acx_word);
close all;clear all;clc;
```

This will create the file:



And paste the active MATLAB figure with a legend into the document:

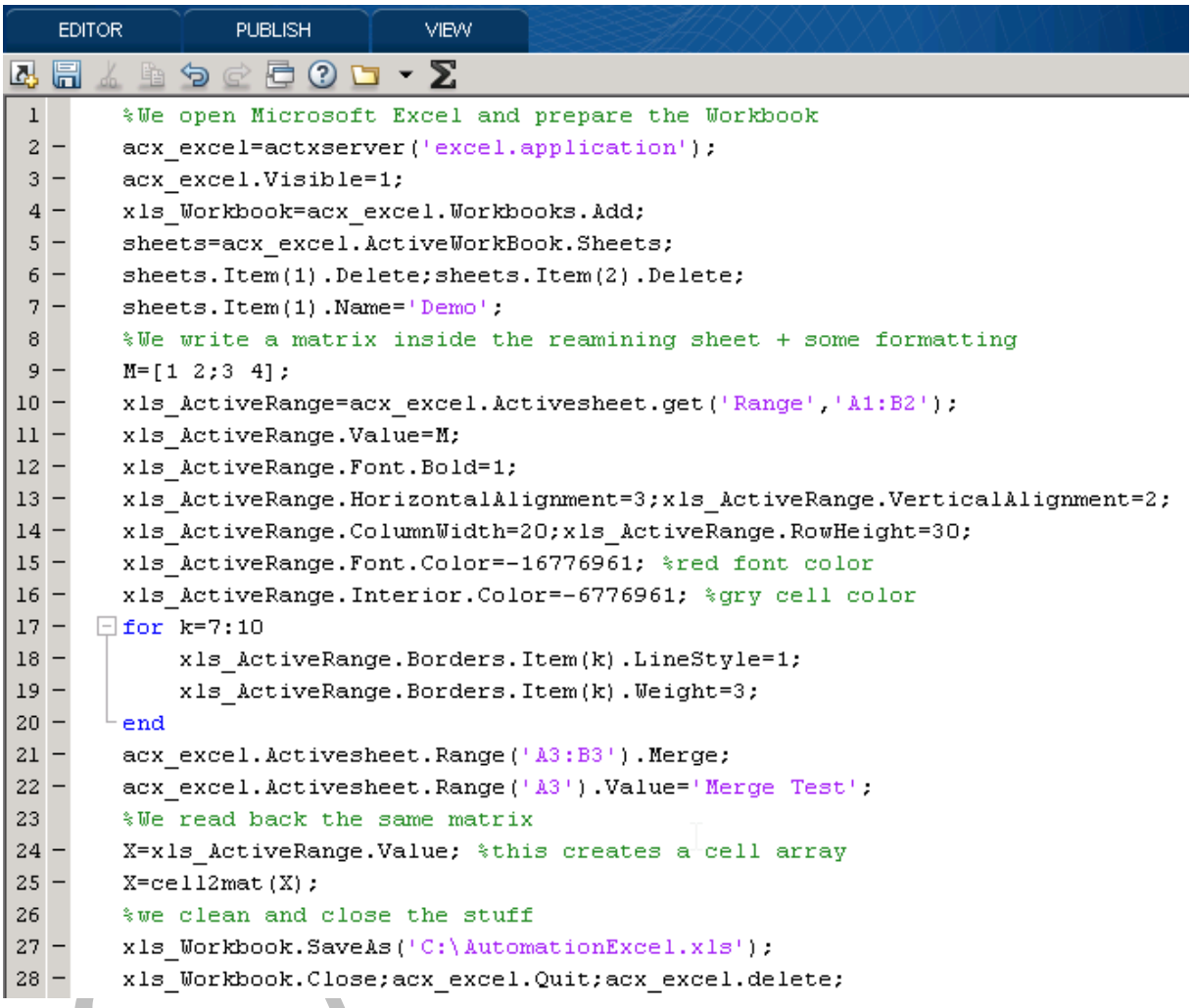


That's it?

Put values into Microsoft Excel

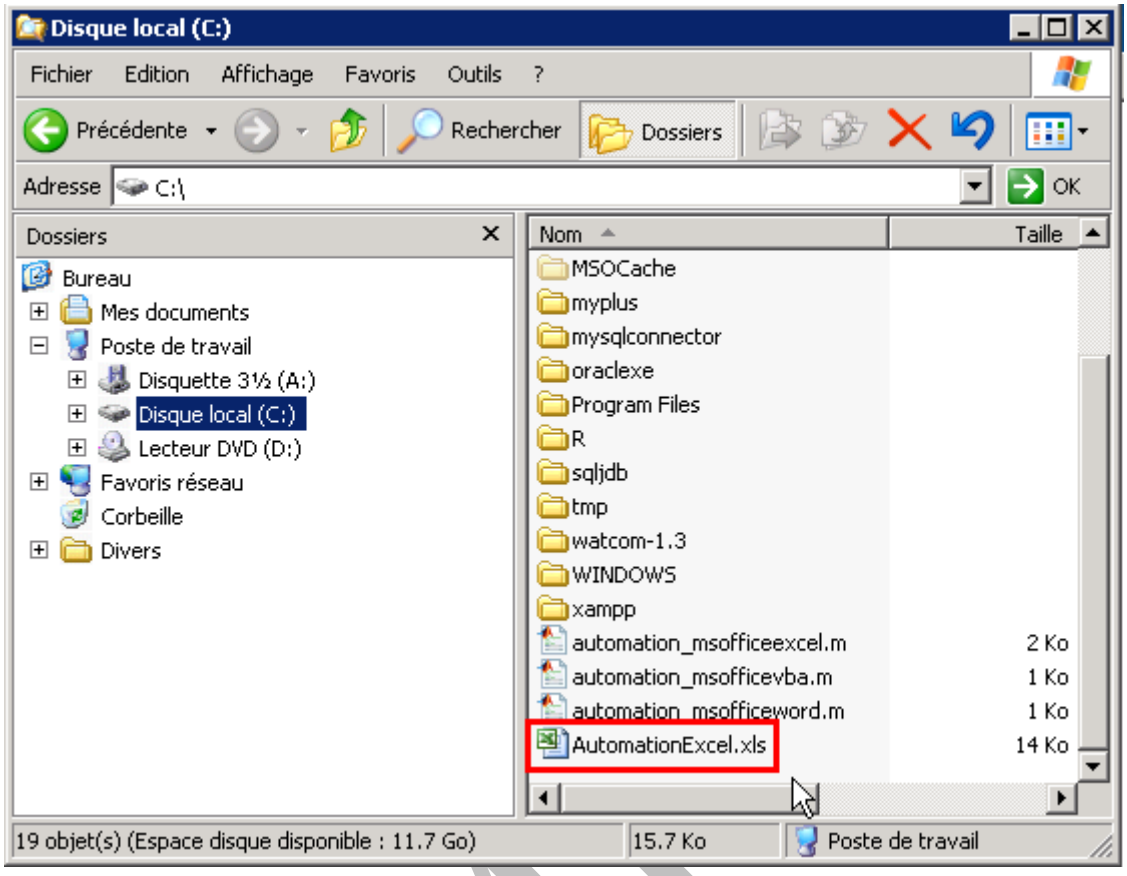
Another very common application of automation is to retrieve MATLAB results into Microsoft Excel.

Here is a Microsoft Excel Automation example with some of the most typical tasks:

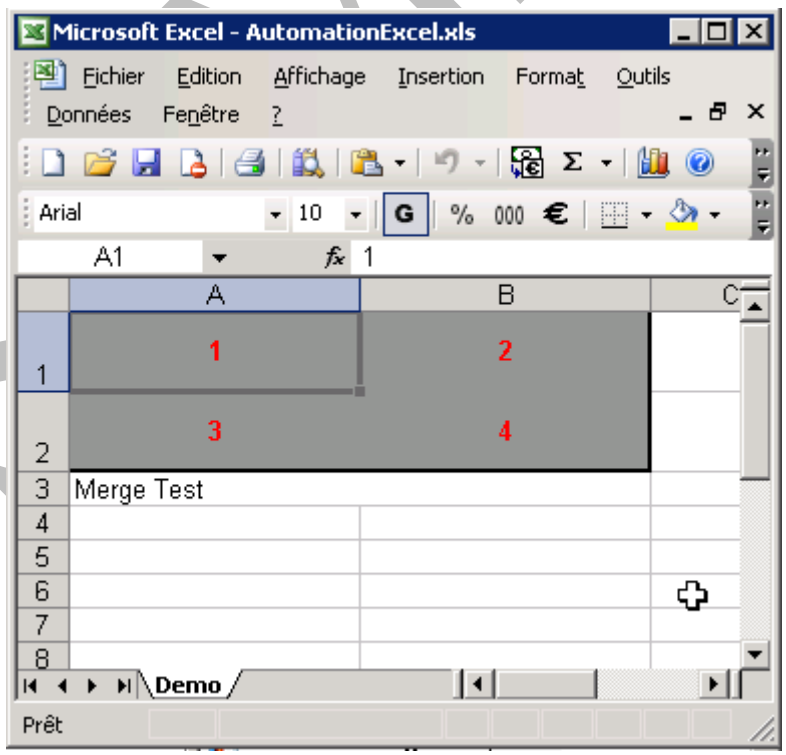



```
%We open Microsoft Excel and prepare the Workbook
acx_excel=actxserver('excel.application');
acx_excel.Visible=1;
xls_Workbook=acx_excel.Workbooks.Add;
sheets=acx_excel.ActiveWorkBook.Sheets;
sheets.Item(1).Delete;sheets.Item(2).Delete;
sheets.Item(1).Name='Demo';
%We write a matrix inside the reamining sheet + some formatting
M=[1 2;3 4];
xls_ActiveRange=acx_excel.Activesheet.get('Range','A1:B2');
xls_ActiveRange.Value=M;
xls_ActiveRange.Font.Bold=1;
xls_ActiveRange.HorizontalAlignment=3;xls_ActiveRange.VerticalAlignment=2;
xls_ActiveRange.ColumnWidth=20;xls_ActiveRange.RowHeight=30;
xls_ActiveRange.Font.Color=-16776961; %red font color
xls_ActiveRange.Interior.Color=-6776961; %gry cell color
for k=7:10
    xls_ActiveRange.Borders.Item(k).LineStyle=1;
    xls_ActiveRange.Borders.Item(k).Weight=3;
end
acx_excel.Activesheet.Range('A3:B3').Merge;
acx_excel.Activesheet.Range('A3').Value='Merge Test';
%We read back the same matrix
X=xls_ActiveRange.Value; %this creates a cell array
X=cell2mat(X);
%we clean and close the stuff
xls_Workbook.SaveAs('C:\AutomationExcel.xls');
xls_Workbook.Close;acx_excel.Quit;acx_excel.delete;
```

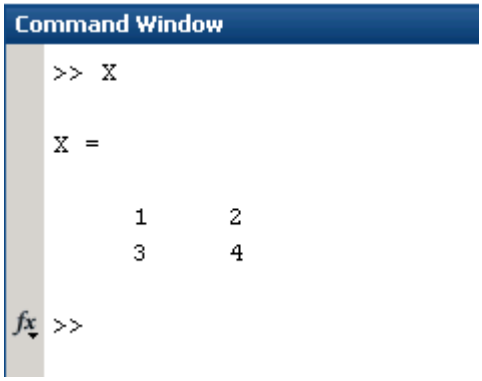
This will create a file as you can see:



With the following content:



And because our example has also read the values that very into the Microsoft Excel sheet to retrieve them into a MATLAB variable X we have:

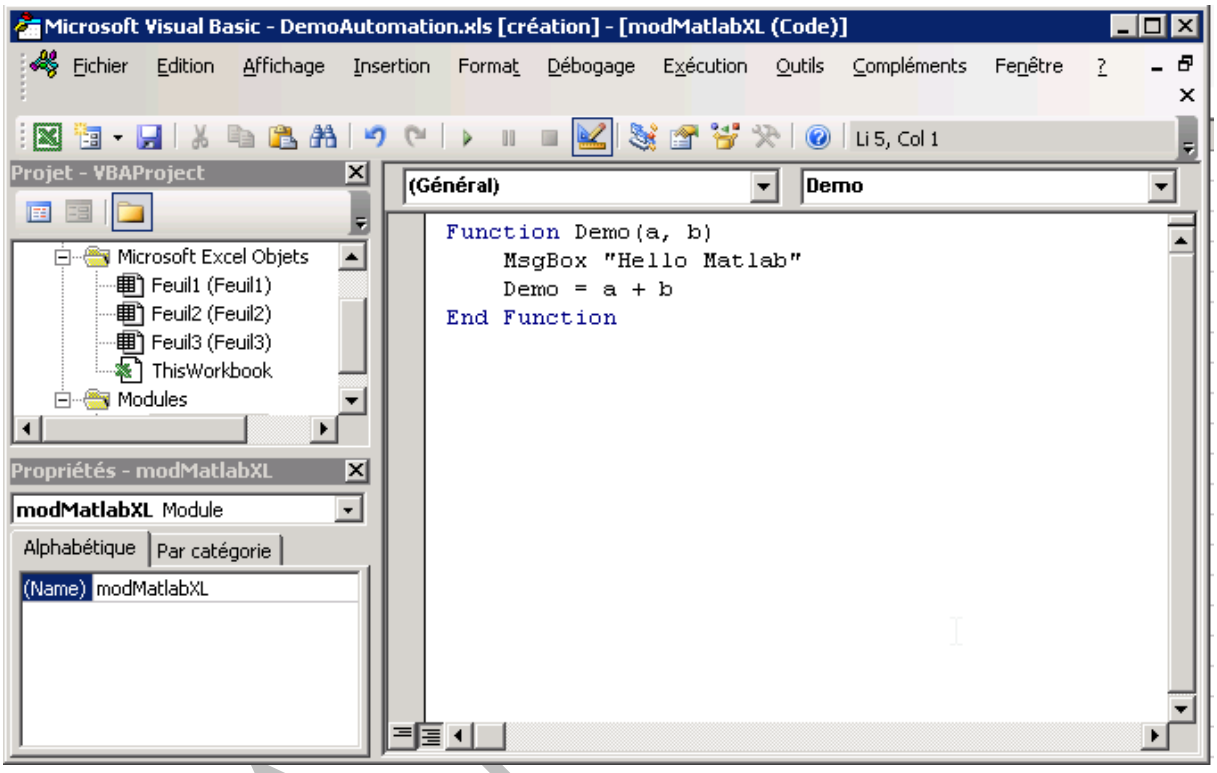


SAMPLE

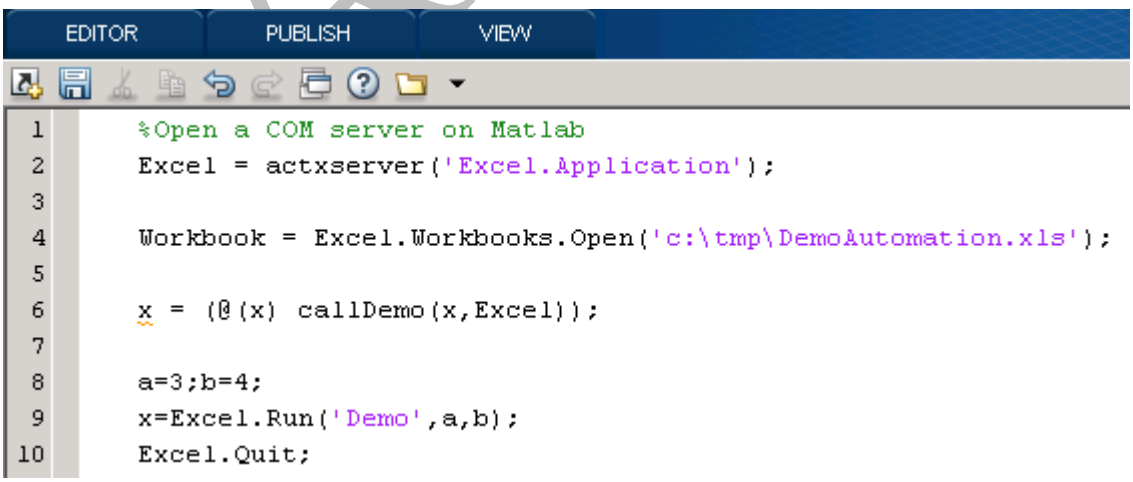
Get values from a Microsoft Excel VBA function

A lot of people develop Microsoft Excel functions and because of time and budget constraints financial engineers can't convert VBA functions to MATLAB script. Then an easy way is to call VBA functions from Microsoft Excel.

Consider the following Excel file named *DemoAutomation.xls*, with the module *modMATLABXL* and a function inside *Demo*:



Now the script to call the VBA function from MATLAB is the following:



If we run this code, we get:

```
%Open a COM server on Matlab
Excel = actxserver('Excel.Application');

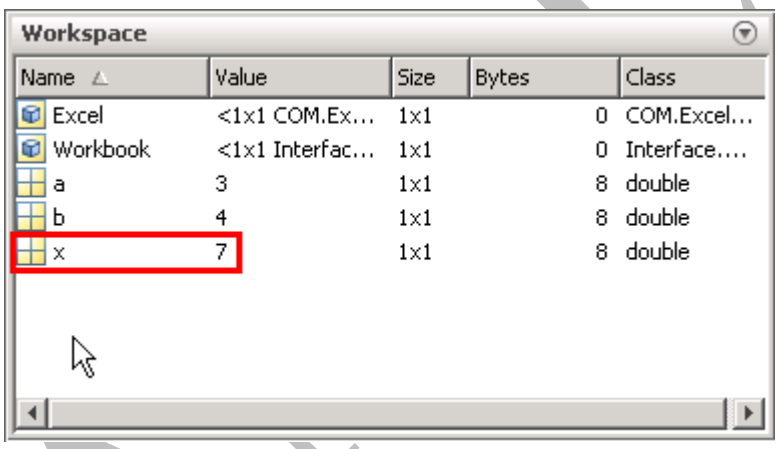
Workbook = Excel.Workbooks.Open('c:\tmp\DemoAutomation.xls');

x = (@(x) callDemo(x,Excel));

a=3;b=4;
x=Excel.Run('Demo',a,b);
Excel.Quit;
```



And in the workspace:



Gotcha!

SAMPLE

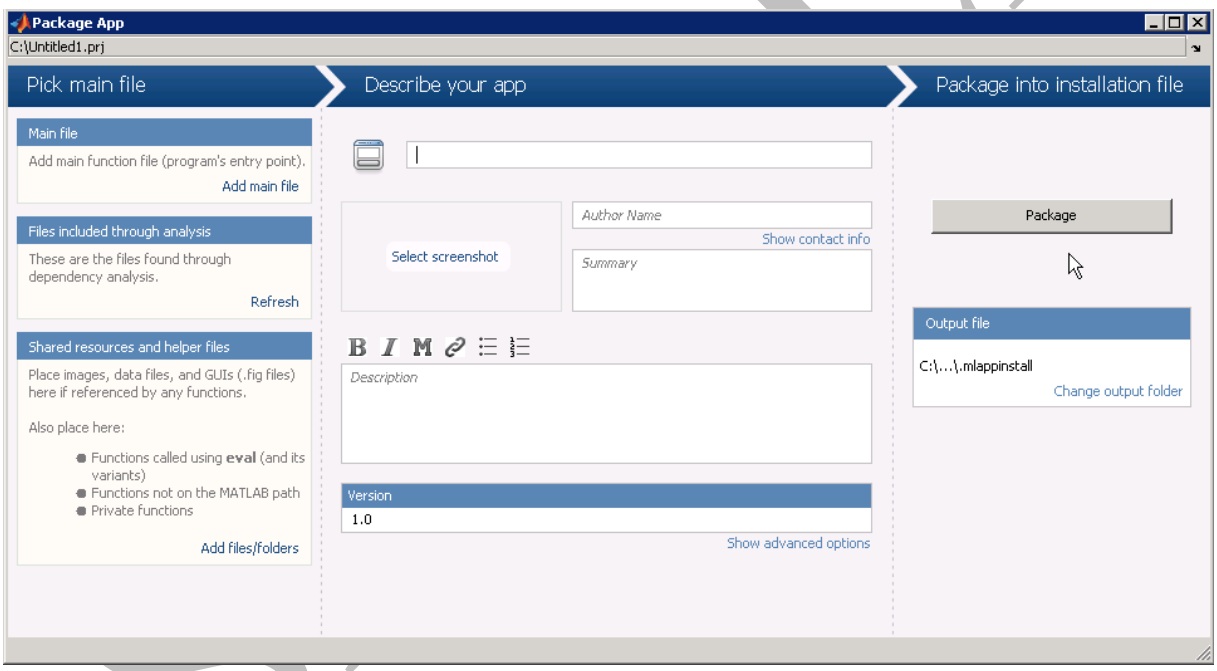
Create a ToolBox (App)

Doesn't matter if you are a professional MATLAB developer or not, is much more elegant to share or sell a complicated MATLAB script as Toolbox (or "App") than as simple multiple *.m script files.

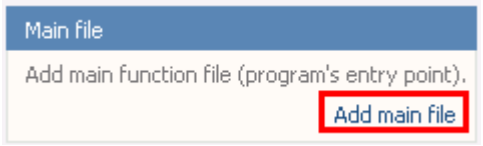
To create a package from your own script you go first then the App ribbons and click on **Package App**:



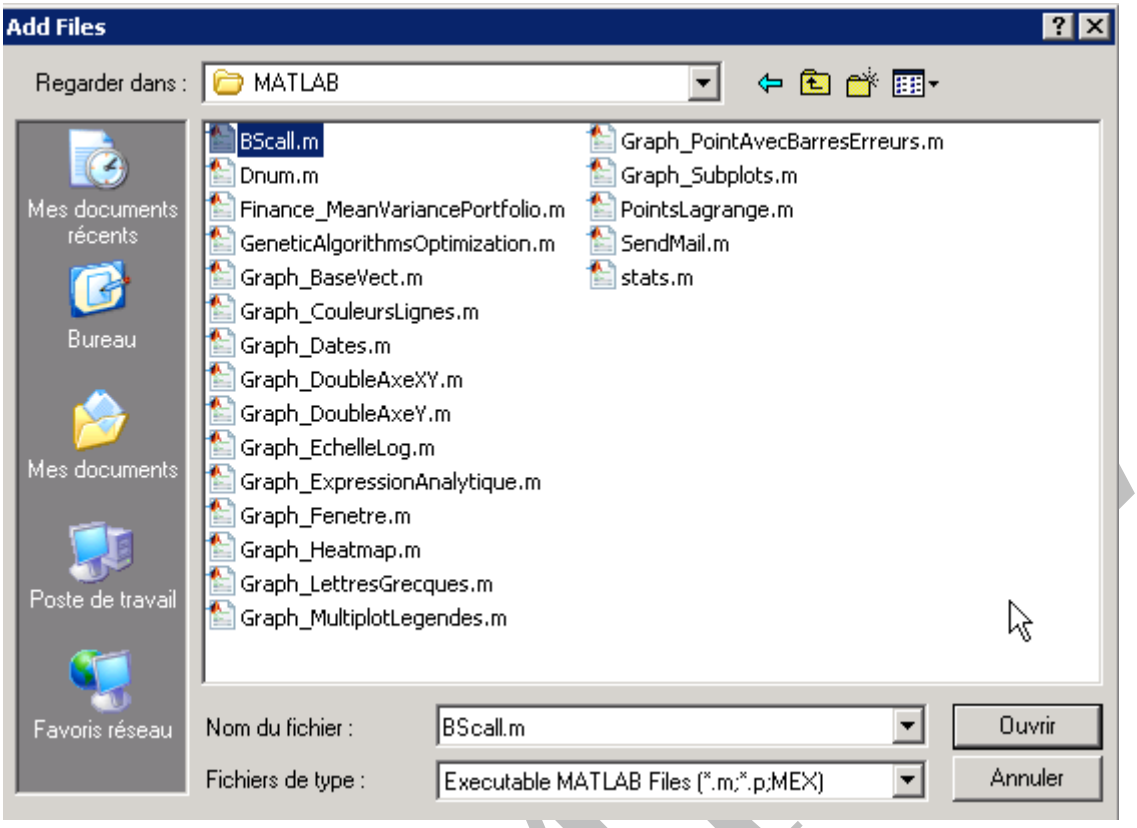
The following wizard will appear on your screen:



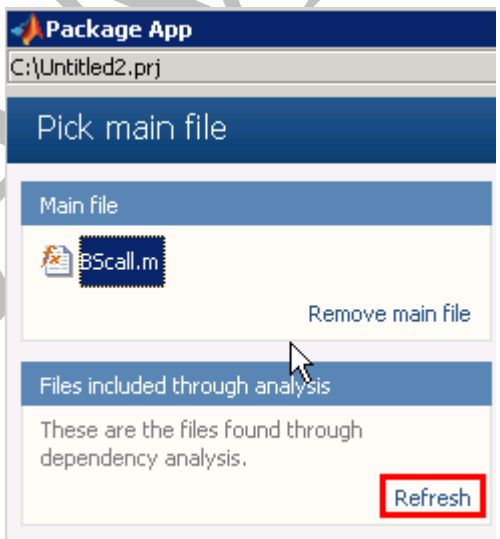
In the upper left corner of the wizard you can see the following. Click on **Add main file**:



Browse for the main file of you script:

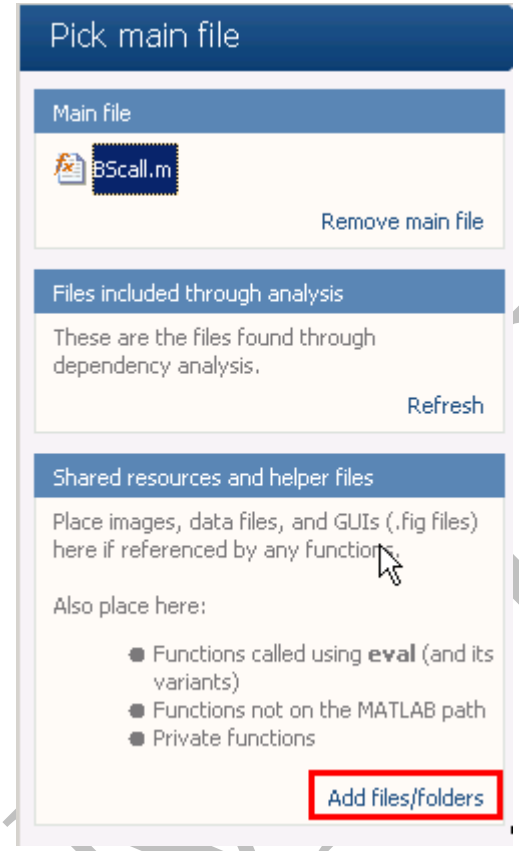


Once this is done you can ask MATLAB to include automatically dependent script files to the main one by clicking on **Refresh**:

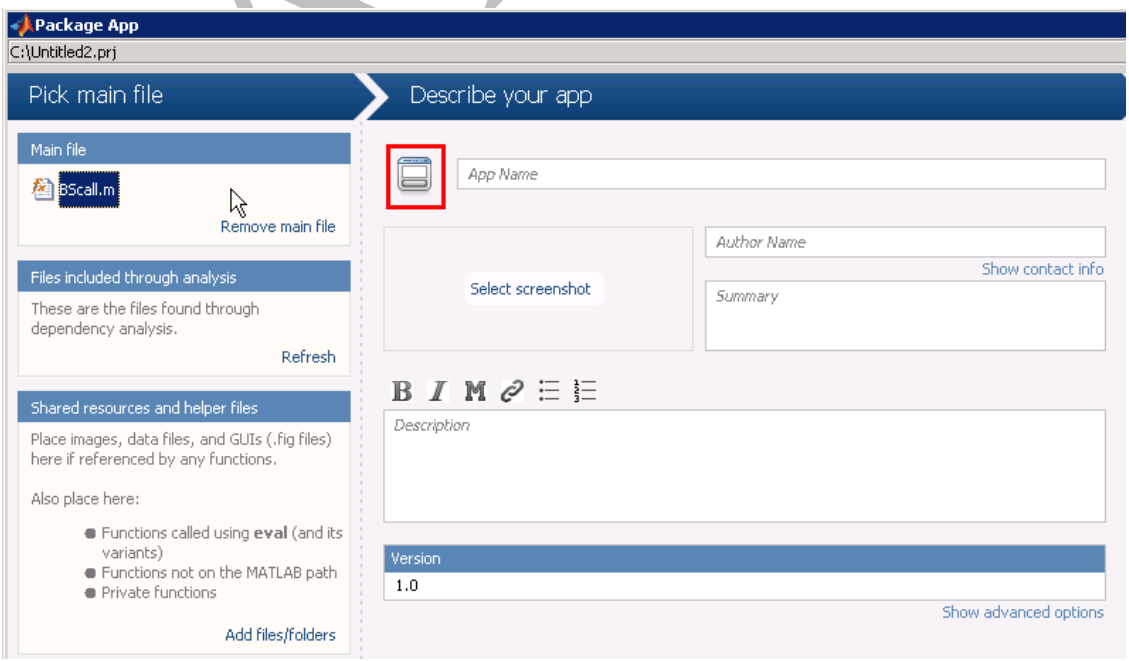


In this case we don't have any dependent files then nothing will appear.

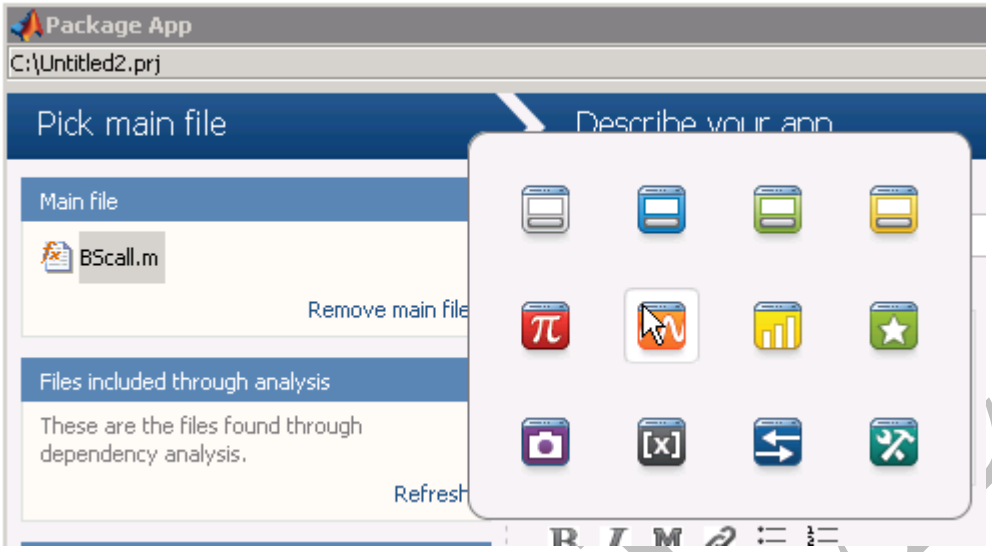
In the third area MATLAB ask you if you have or not any images, data files or GUIs that you would like to associate with you Toolbox:



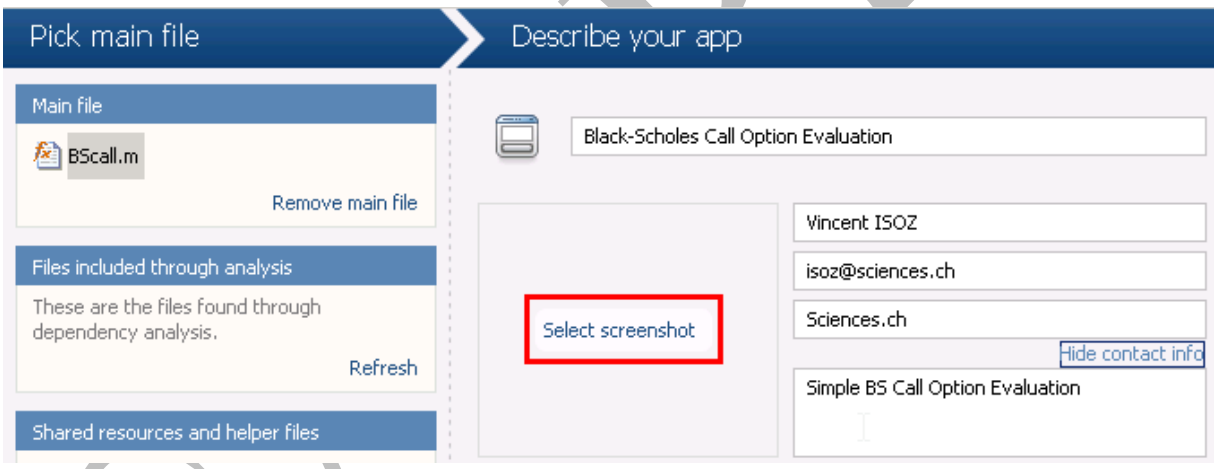
Once the first step finish if you click now on the small icon visible below and rounded by a red rectangle:



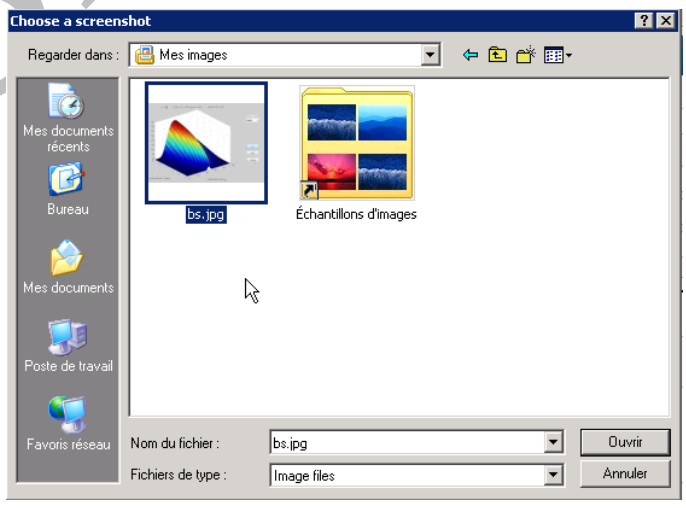
MATLAB ask you which icon you want to use for your application (you can't create your own icon with the wizard):



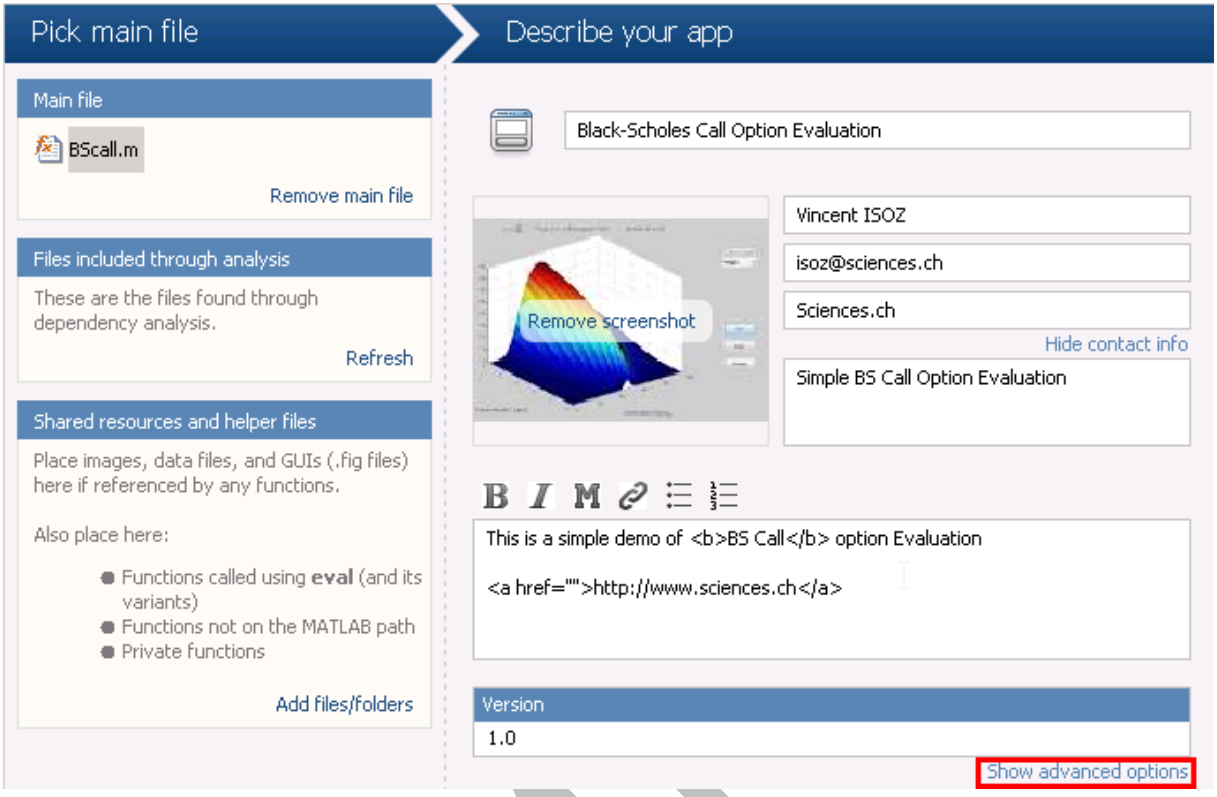
After this you type a title for your App (Toolbox), write some informations about you and you App (Toolbox) and also select a screenshot as preview example:



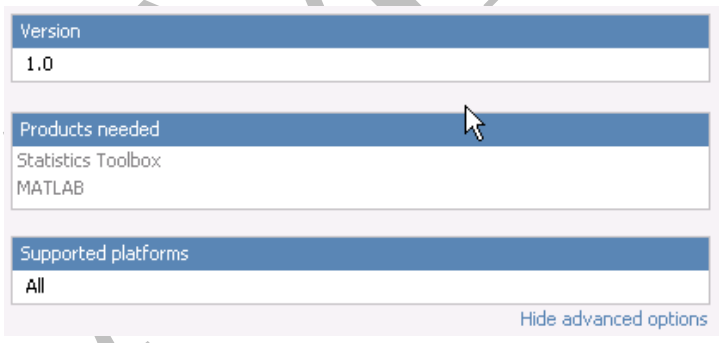
If we click on **Select screenshot**, we get:



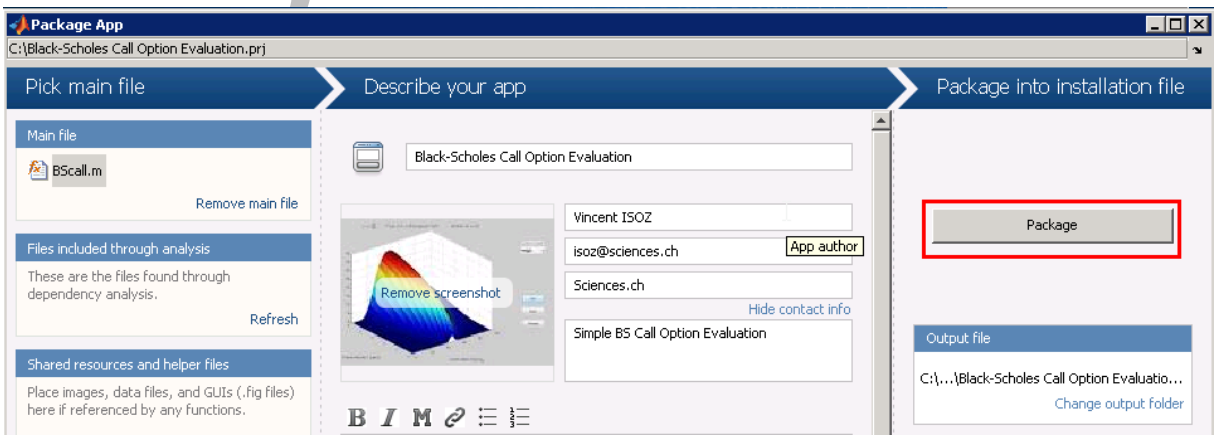
Once the image selected you get:



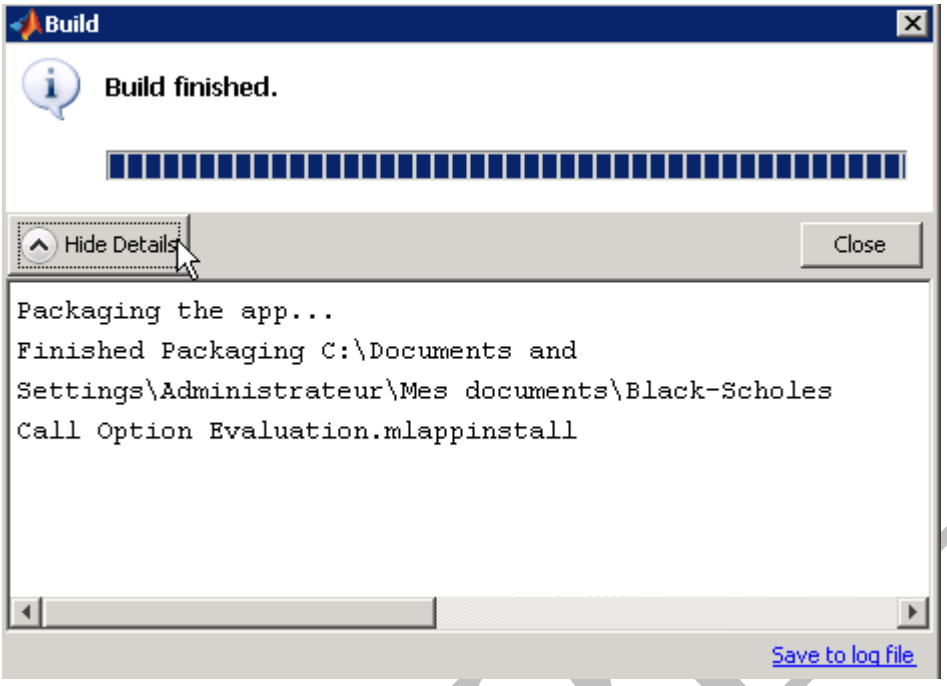
If you click on **Show advanced options** you get:



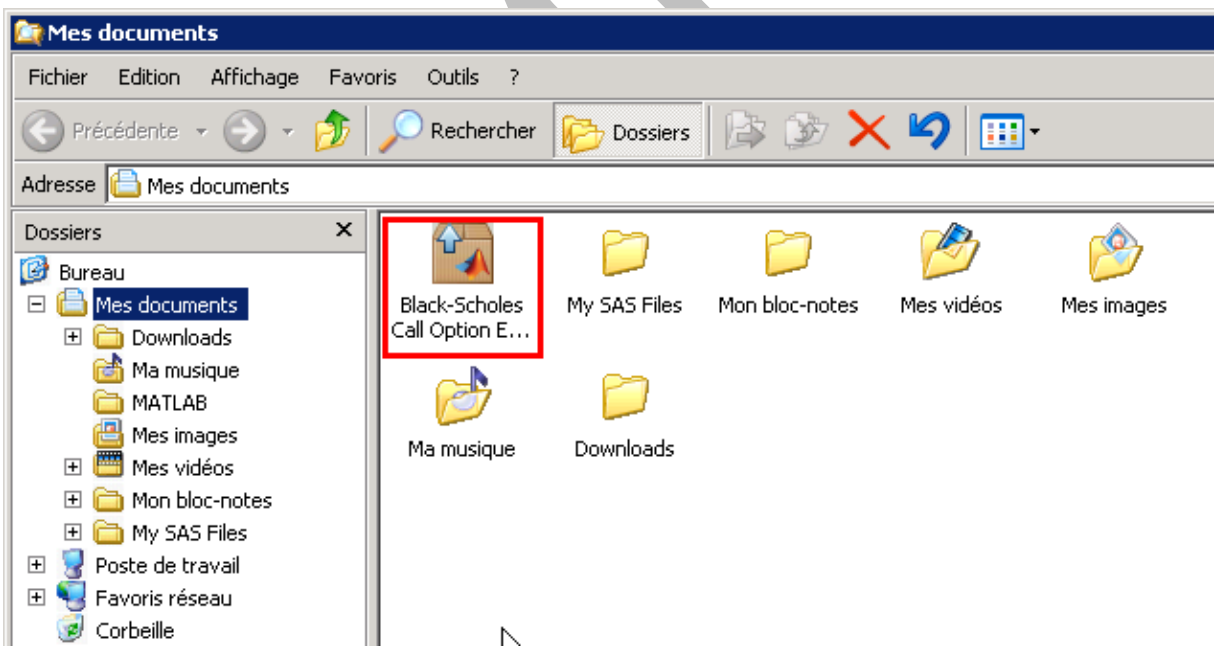
To finish, we click on the button **Package**:



Then the builder opens:

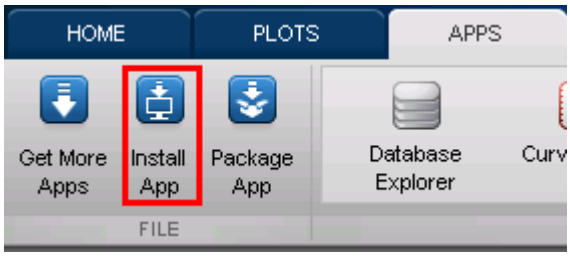


This will create an App in the current workpath folder:

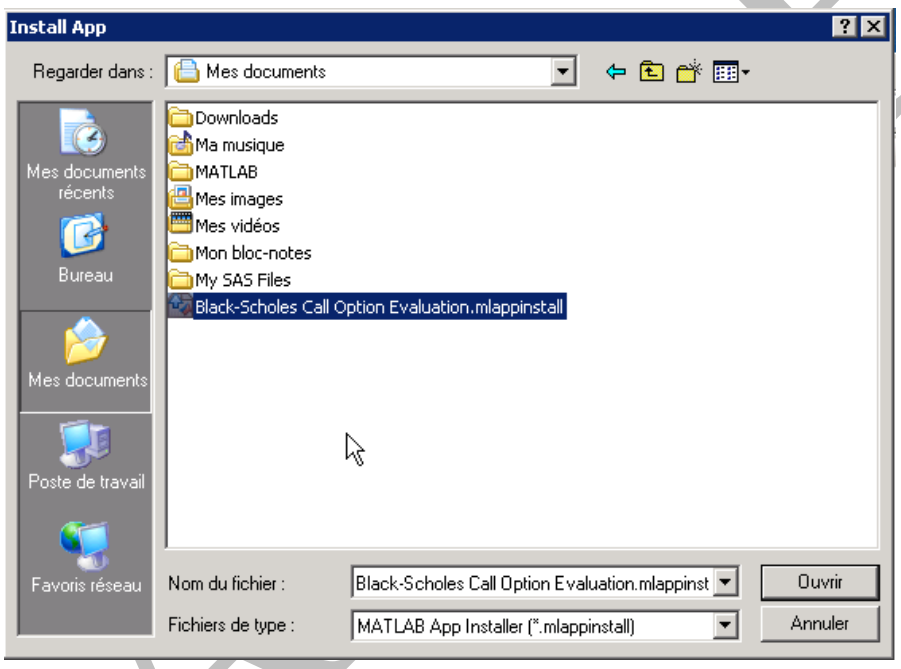


Install/Uninstall a Toolbox

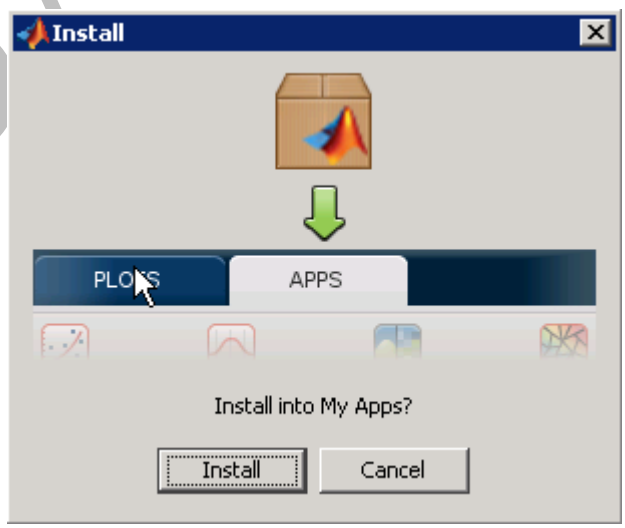
Now once you know how to create an App (Toolbox) you must also know how to install a new App:



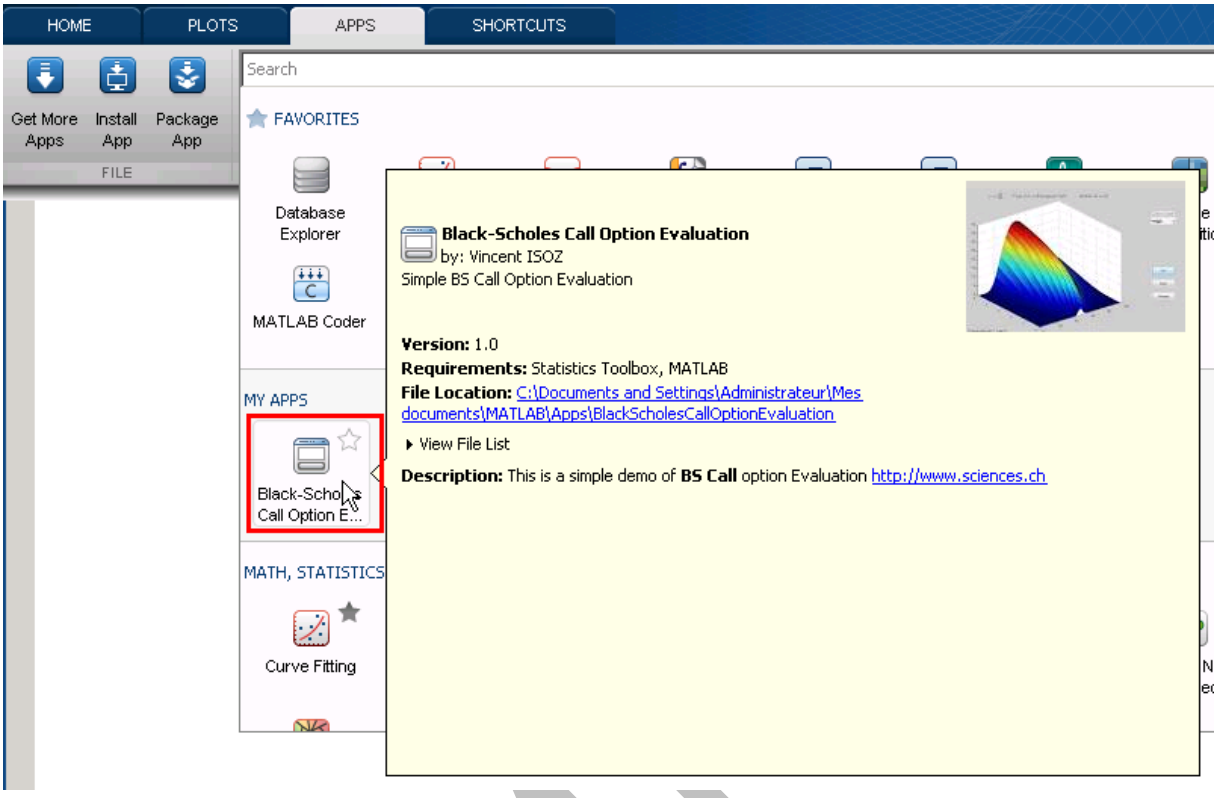
You have to browse where you *.mlappinstall file resides:



Once you select the app the following screen appears:



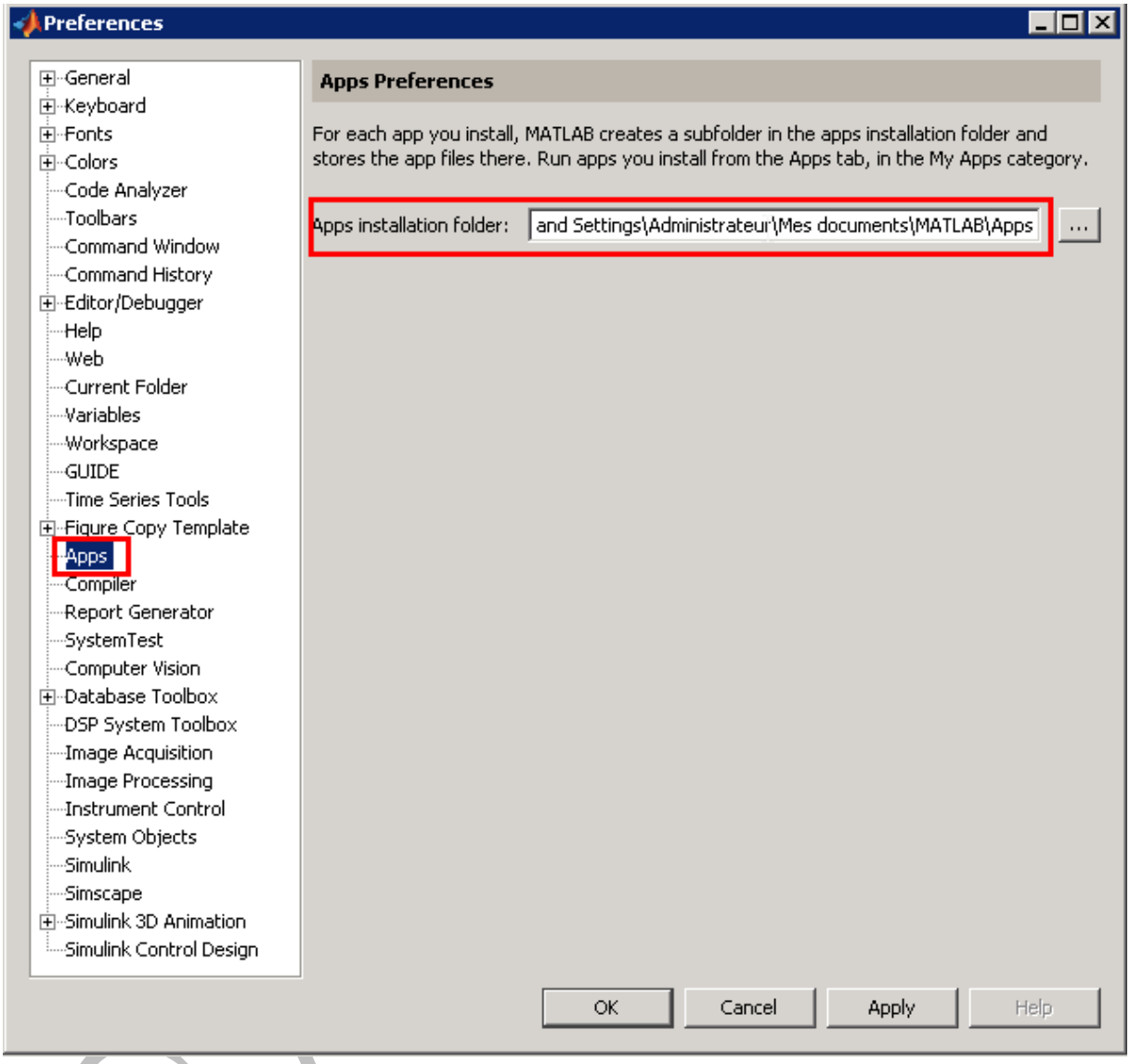
Once the installation finished, you get:



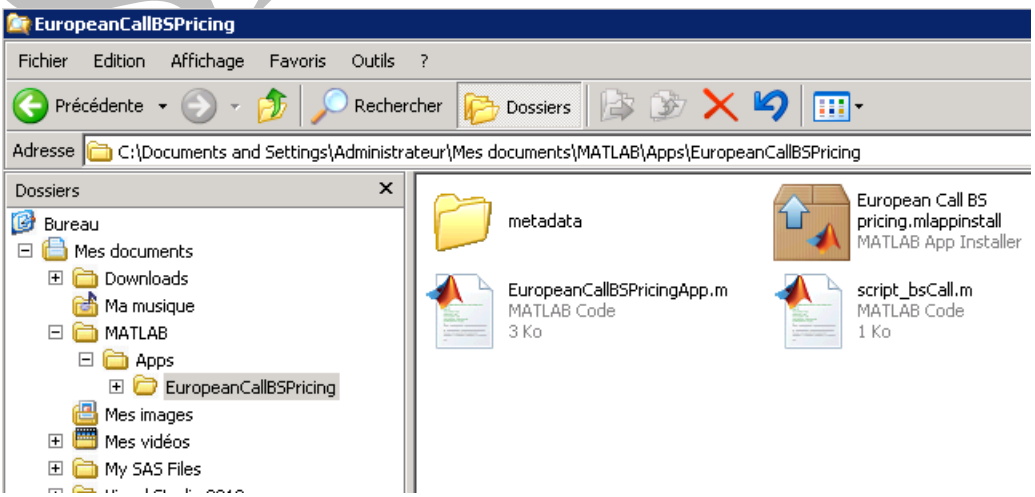
Or an alternative to install is to make a right click on the package and select **Install**:



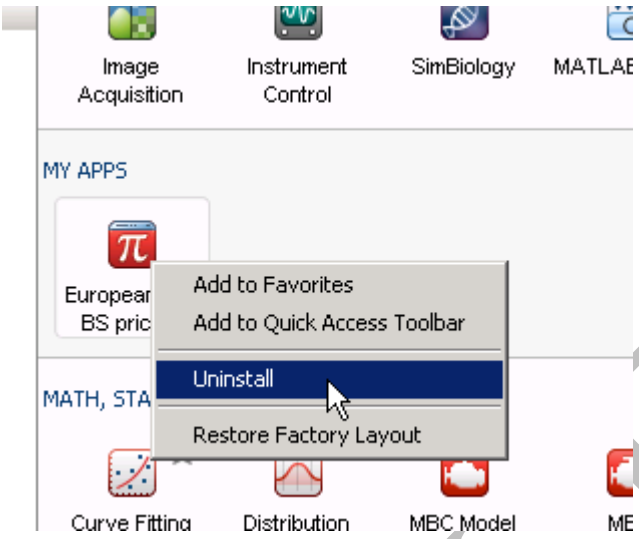
If you want to change the installation folder you just have to go trough the **Preferences** of the software:



And this is how the installation folder looks like:



Uninstall a Toolbox is quite easy:



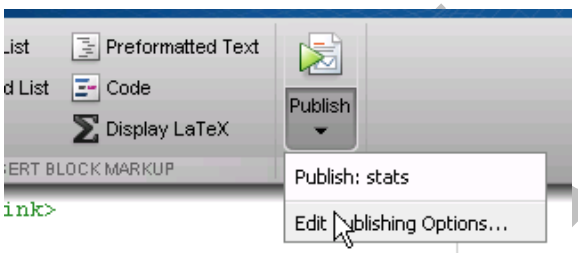
SAMPLE

Publish a script (documented script)

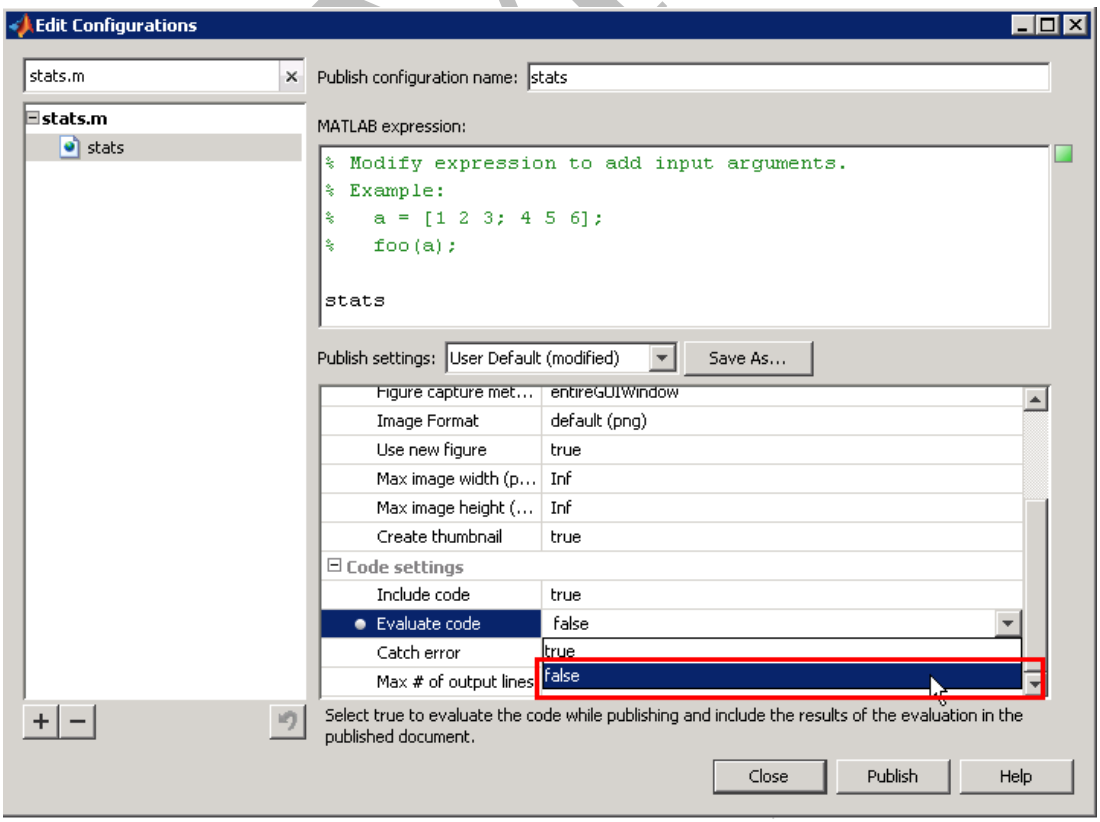
Once you have finished to write a script, an obligation in some industrials fields is to create a documentation of your script. There are some interesting options in the register **PUBLISH** for this purpose:



To publish the file you click on the button Publish and I strongly recommend you to edit some properties first:



One important function is the **Evaluate code** property because if you let this option on **true** (default value) then the whole script will be run each time you publish your code:



Then I strongly recommend first to change this property to false.

After as example considers the following script (see that there is a link and also a LaTex equation!):

```
%% SECTION TITLE
% DESCRIPTIVE TEXT
% Ceci est un texte en *gras*
%
% Example:
%
% Type publish_script(X) to get...
% See also: MEAN
%
% Pour plus d'infos voir: <http://www.alphorm.com ALPHORM>

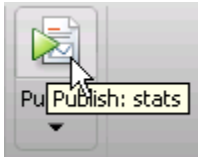
function [mu,sigma]= publish_script(X)
%% Fonction d'exemple simpliste
% Calcule moyenne arith. et écart-type
%%
% * Calcule la moyenne arithmétique
% * Calcule l'écart-type
%
% Rappel de la formule de la moyenne arithmétique
%
%%
% $$\mu= \frac{1}{n} \sum x_i$$
%
% nous utiliserons la fonction MEAN native de MATLAB

    %% section  moyenne arithmétique
    % classique...

    mu=mean(X);

    %% section écart-type
    % classique aussi...
    if nargout==2
        [nobs,nc]=size(X);
        if nobs==1, nobs=nc; end
        sigma=sqrt(sum(X.^2)/nobs-mu.^2);
    end
    %% exemple d'insertion d'un graphique
    % <<peaks.jpg>>
end
```

Now if we **Publish** the code:



The result will look like following:



Contents

- SECTION TITLE
- Fonction d'exemple simpliste
- section moyenne arithmétique
- section écart-type
- exemple d'insertion d'un graphique

SECTION TITLE

DESCRIPTIVE TEXT Ceci est un texte en **gras**

Example:

Type publish_script(X) to get... See also: MEAN

Pour plus d'infos voir: ALPHORM

```
function [mu,sigma]= publish_script(X)
```

Fonction d'exemple simpliste

Calcule moyenne arith. et écart-type

- Calcule la moyenne arithmétique
- Calcule l'écart-type

Rappel de la formule de la moyenne arithmétique

$$\mu = \frac{1}{n}\sum x_i$$

nous utiliserons la fonction MEAN native de MATLAB

section moyenne arithmétique

classique...

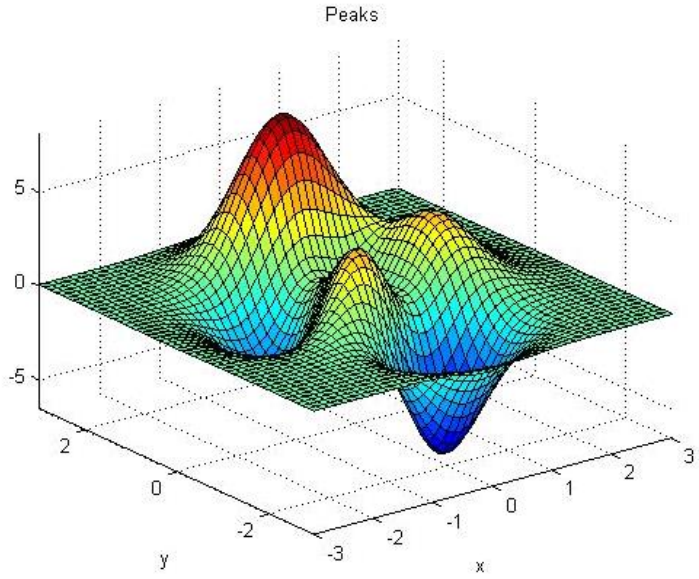
```
    mu=mean(X);
```

section écart-type

classique aussi...

```
    if nargout==2
        [nobs,nc]=size(X);
        if nobs==1, nobs=nc; end
        sigma=sqrt(sum(X.^2)/nobs-mu.^2);
    end
```

exemple d'insertion d'un graphique

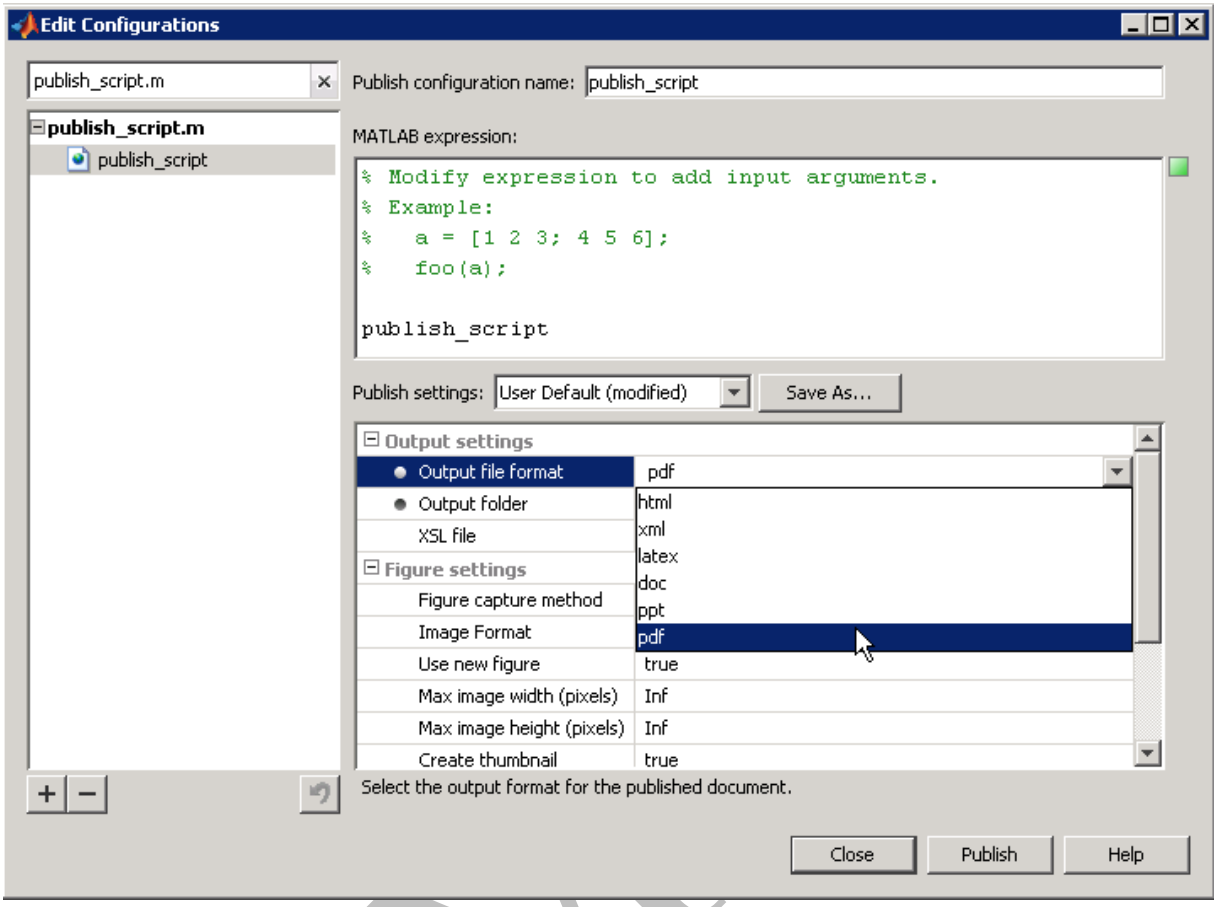



```
end
```

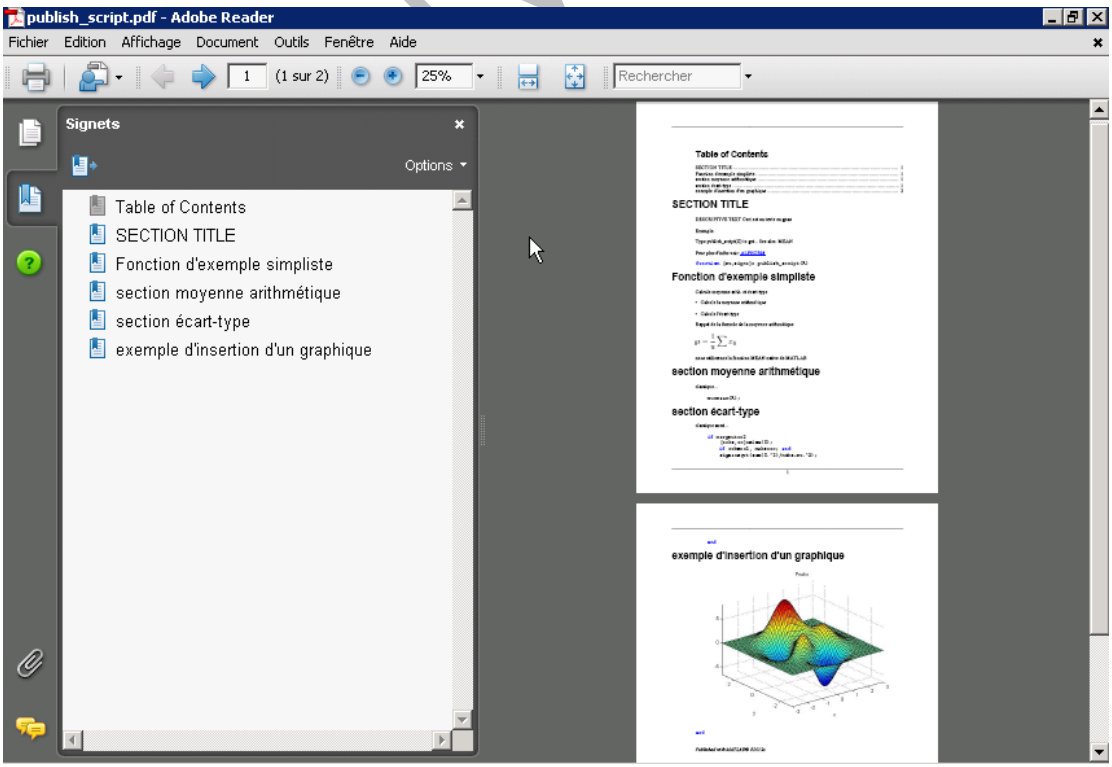
Published with MATLAB® R2013a

SAMPLE

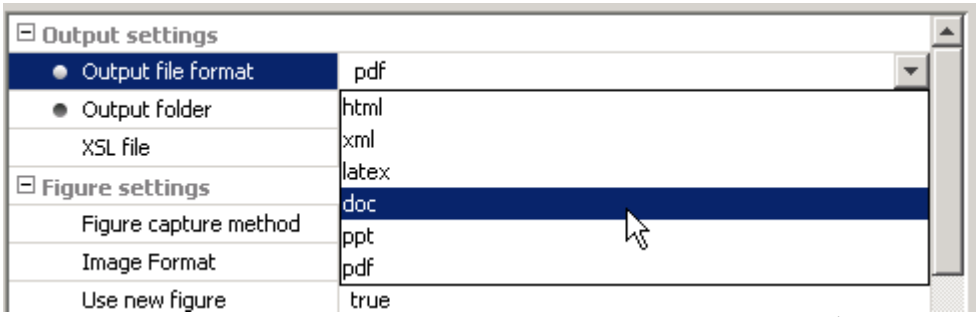
If we change the output option to a PDF:



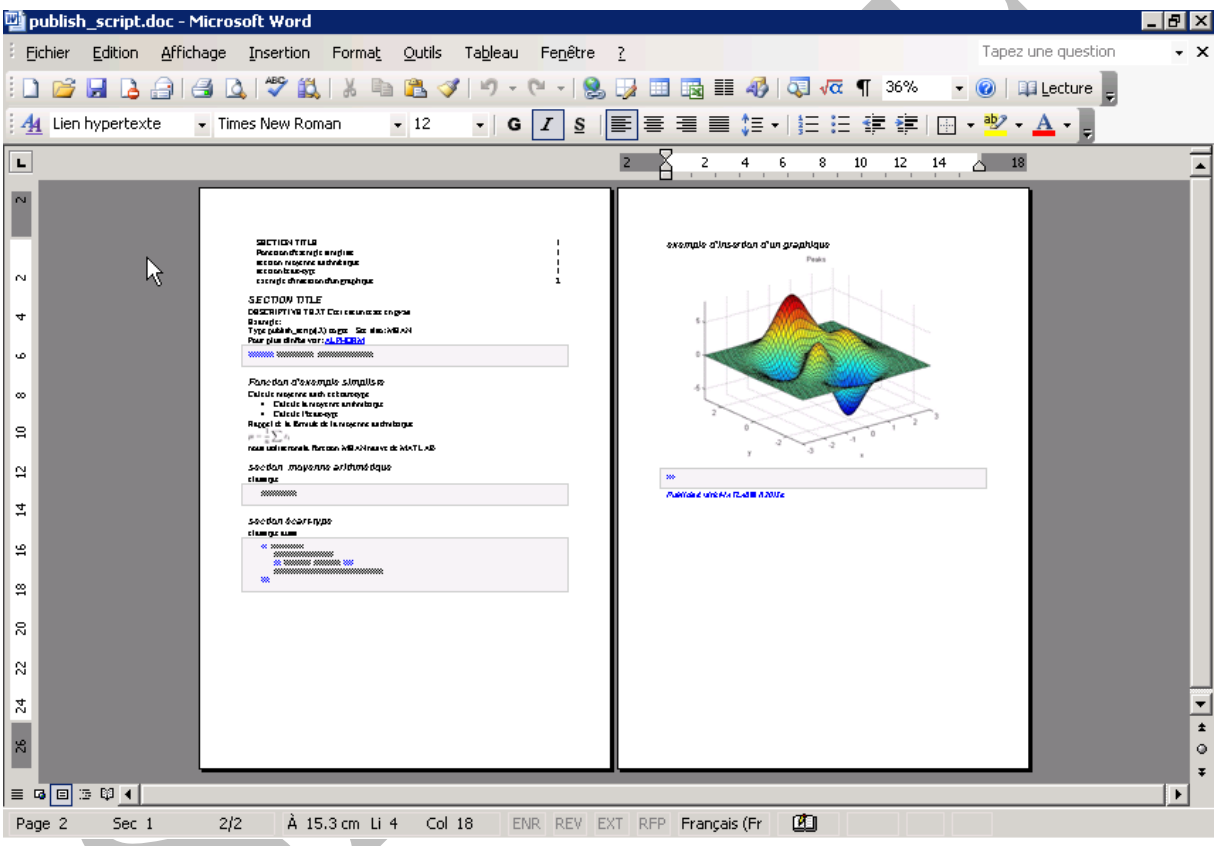
We get a very clean result with a table of content and clickable links:



And if we change the output to *.doc:



we also get a very clean output with a real table of contents, styles, clickable links, etc.:

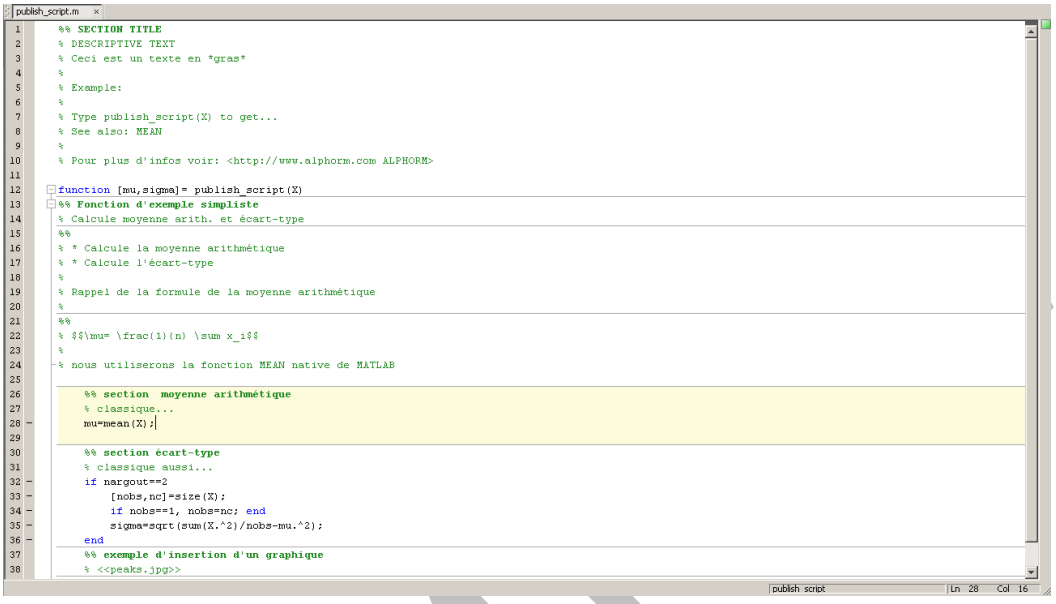


Caution!!! If you choose to run the code in the **Publish** feature property, you will see only the charts at the end of the publication that were still open after the execution finished (you won't see closed charts!).

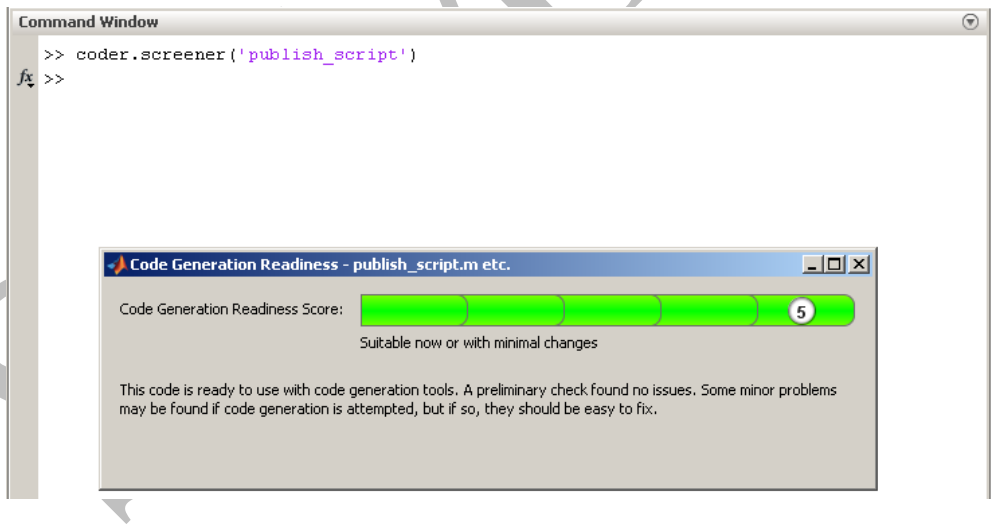
Check Readiness

Check readiness is very important in practice for big applications. Mathworks has developed a tool to check code readiness and that gives a score back.

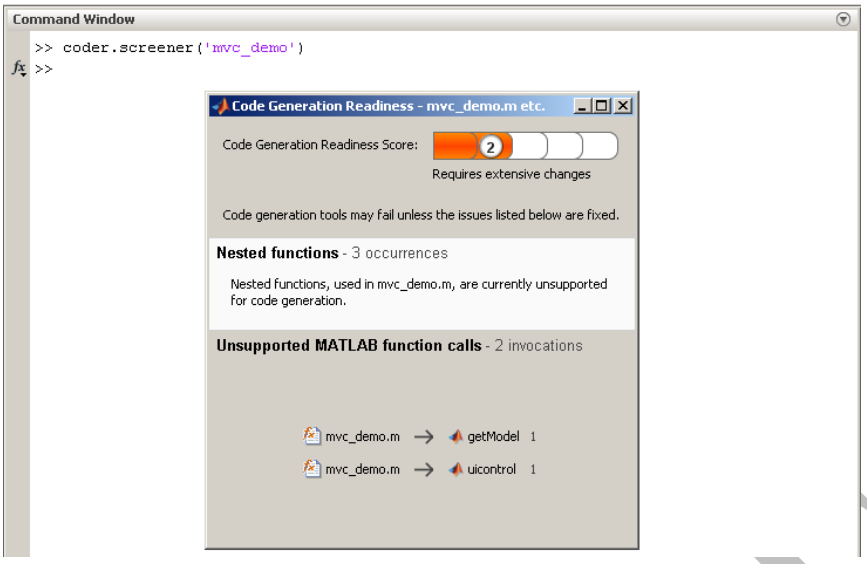
Consider the following script we used just before:



To check its readiness we just use the function **coder.screener( ):**



If the code is not so good we get:


Command Window
>> coder.screener('mvc_demo')
>>
Code Generation Readiness - mvc_demo.m etc.
Code Generation Readiness Score:
2
Requires extensive changes
Code generation tools may fail unless the issues listed below are fixed.
Nested functions - 3 occurrences
Nested functions, used in mvc_demo.m, are currently unsupported for code generation.
Unsupported MATLAB function calls - 2 invocations
mvc_demo.m
getModel 1
mvc_demo.m
uicontrol 1

SAMPLE

Optimization tricks

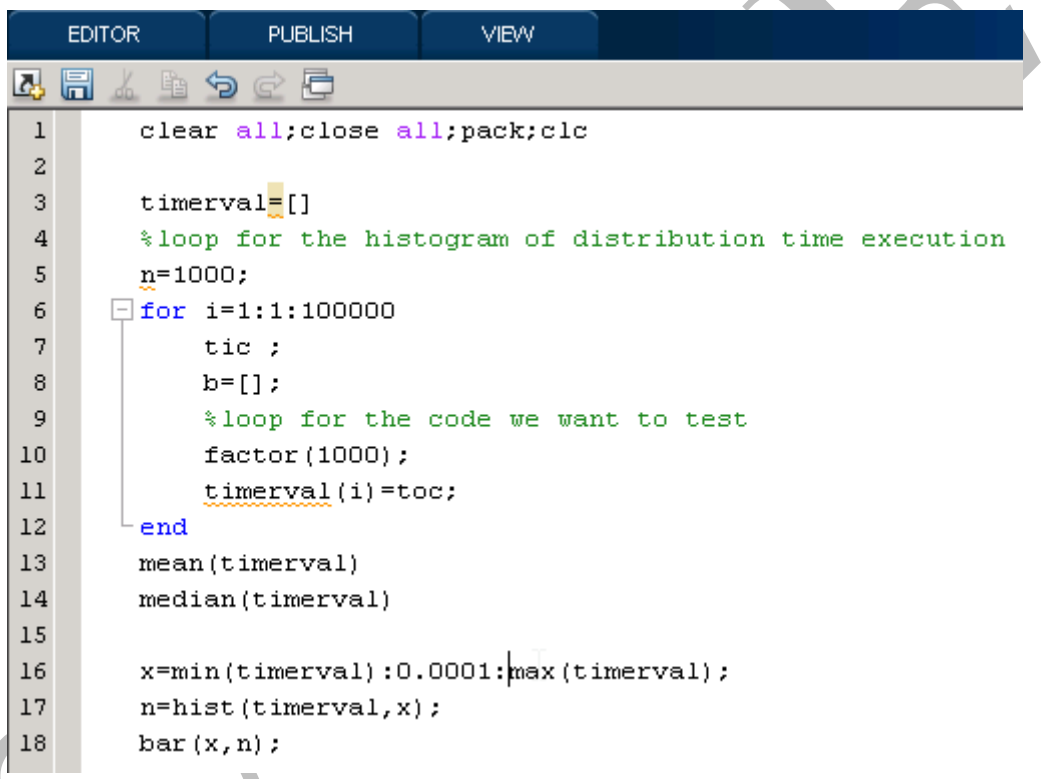
In advanced finance (hyperfrequency trading) and spatial applications, the performance of a script a major factor (with the robustness...).

The purpose of this small chapter is to show you some simple optimization tricks.

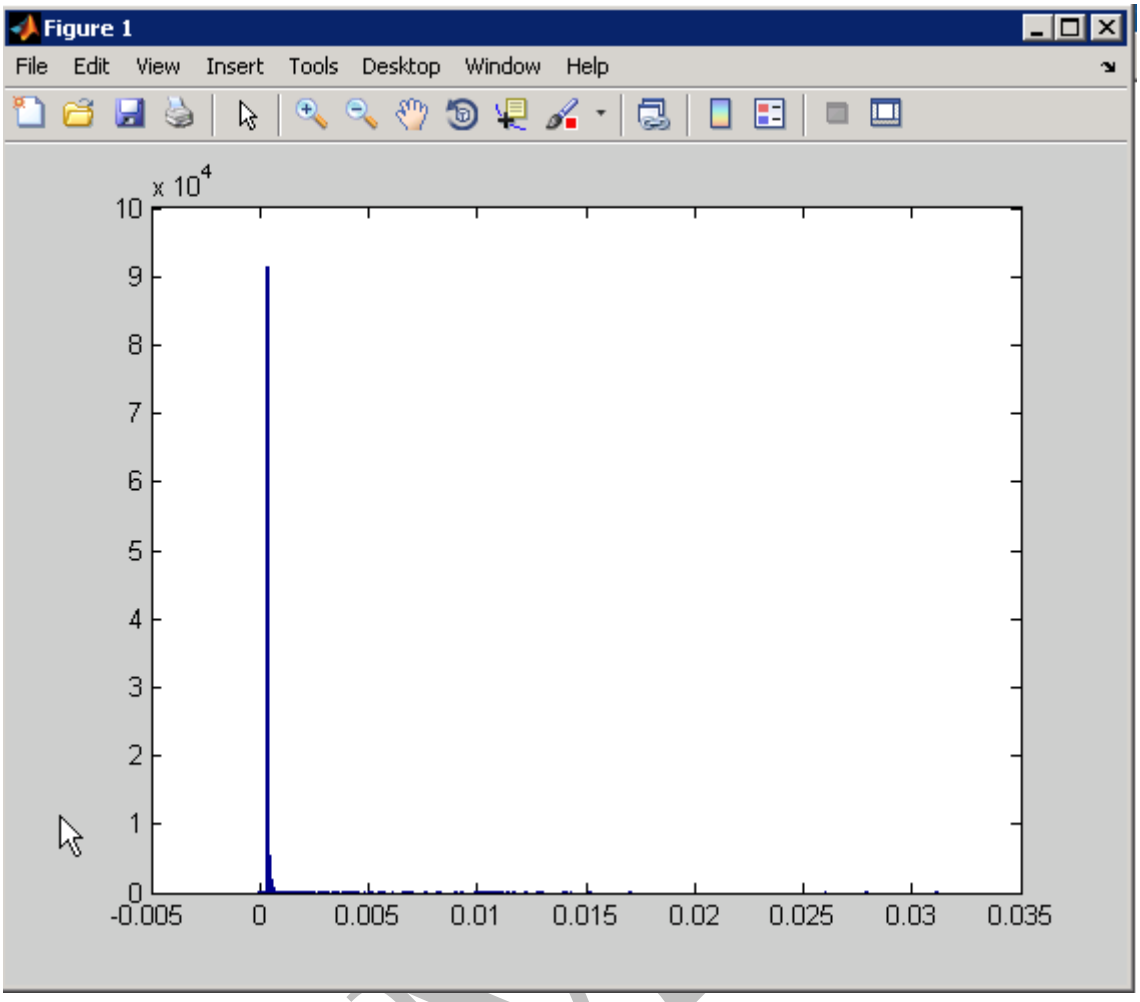
Statistically check code efficiency

When you test the efficiency of a script you must not run it just one time, but thousands of times to get a statistical distribution of running time.

Here is such an example with the famous example (in cryptography) of the **factor( )** function:



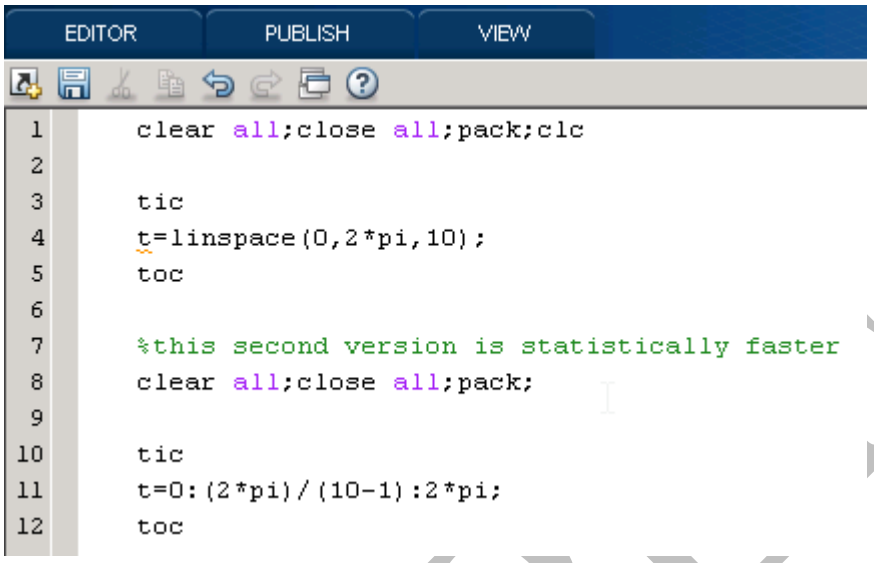
The result will looks like following:


Figure 1
File Edit View Insert Tools Desktop Window Help
x 10^4
10
9
8
7
6
5
4
3
2
1
0
-0.005
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035

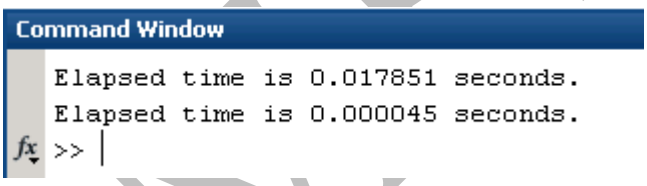
SAM

linspace vs sequence

A lot of people use the **linspace( )** function to create sequence of values but in the facts this **function** is a little bit slow as you can see below:



The running time differences gives:

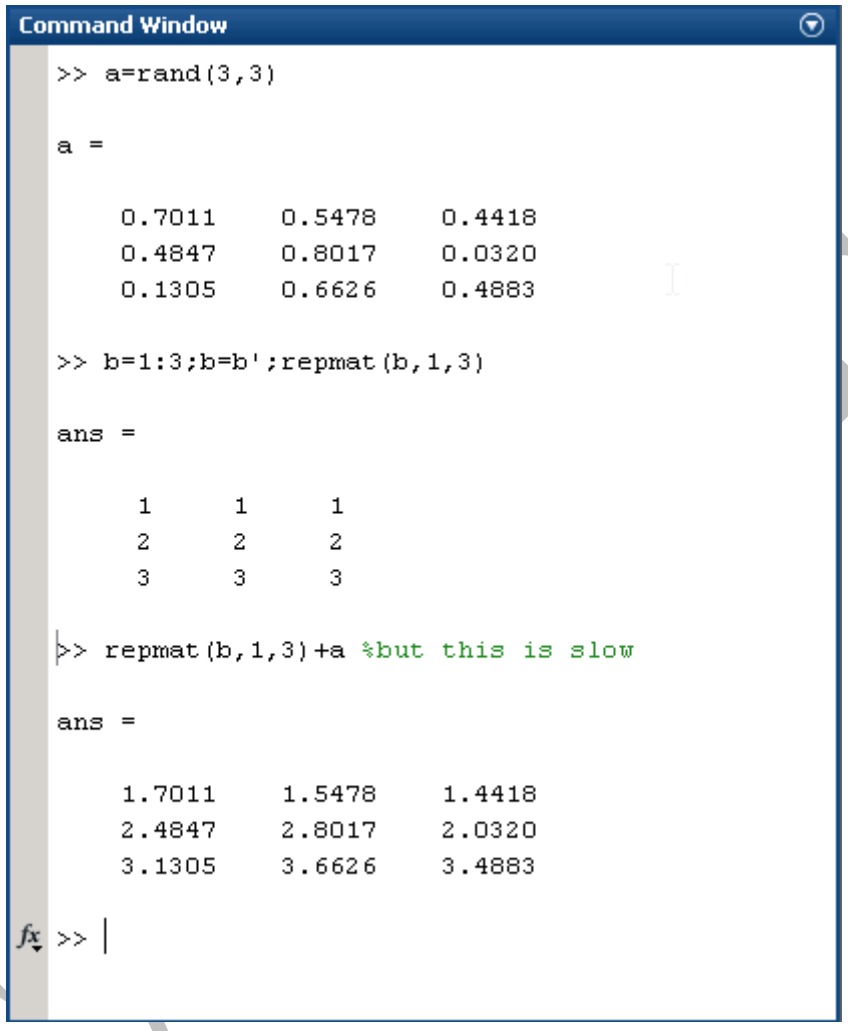


SAMPLE

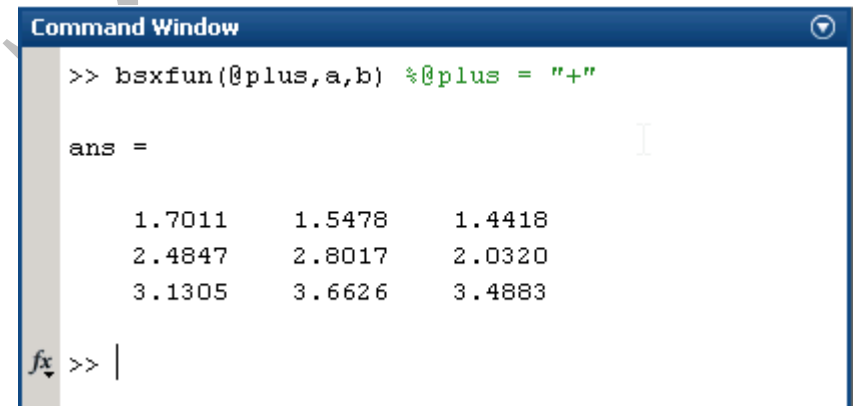
Simple arithmetic operation vs bsxfun

Another difficulty trick to know is to use the broadcasting **bsxfun( )** instead of simple arithmetic operation (and some others) in MATLAB.

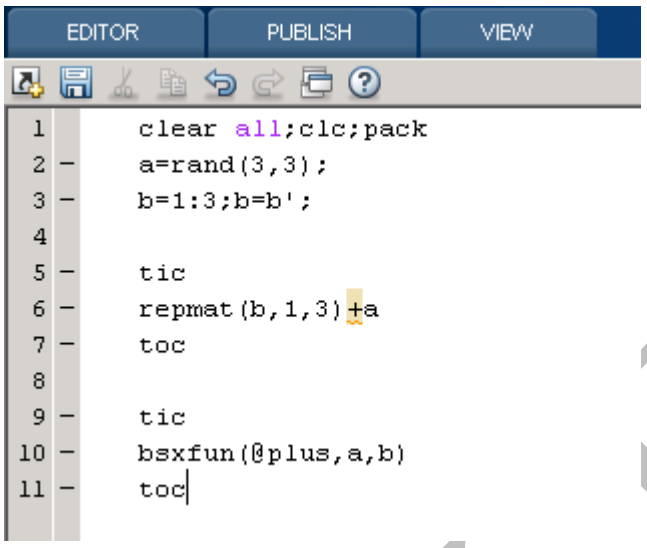
To understand the subject first see and read the following example:



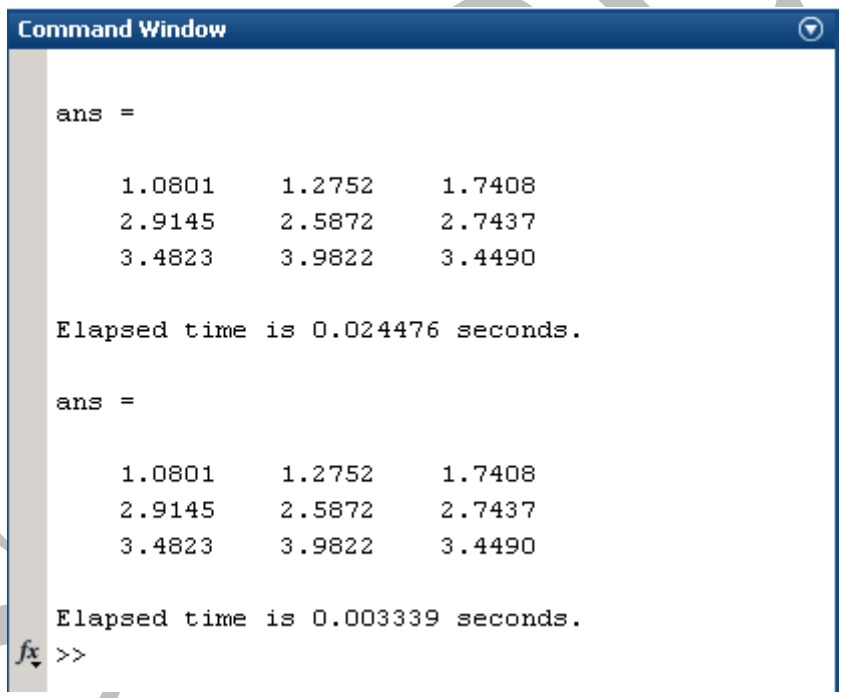
But this is much faster:



Let us now compare the performance between a classic addition and a sum using **bsxfun**:



The result will be:



The most used operators of bsxfun are @plus, @minus, @times, @power, @max, @min.

Create userforms (GUI)

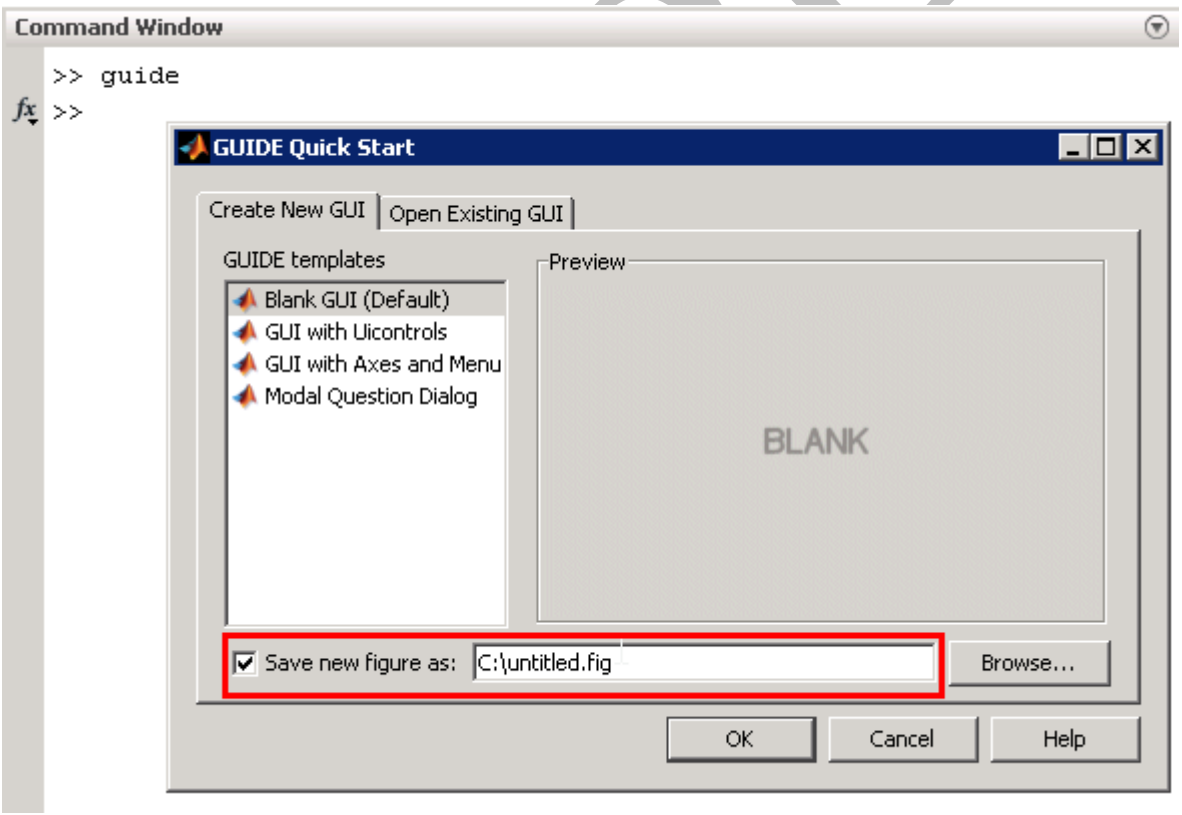
Like coding (doesn't matter which language) non-mathematical stuff is very boring in my point of view, create GUI is also a much more boring job that could better be given to a student or a designer rather than a senior scientist having a PhD or a Master or even an undergraduate level.

GUI have a lot of possibilities like in almost all scripting/programming languages this is why this section/chapter about GUI will be written very slowly and perhaps never completed. Hope you understand what I mean...

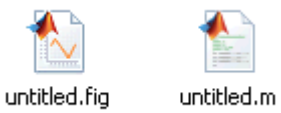
Create userforms using GUIDE

Create GUI in MATLAB is a huge subject! There a 300 hundred pages book only about this tool! The purpose of this PDF will not be to make you a professional of GUI creation but just to give you an idea of how it works.

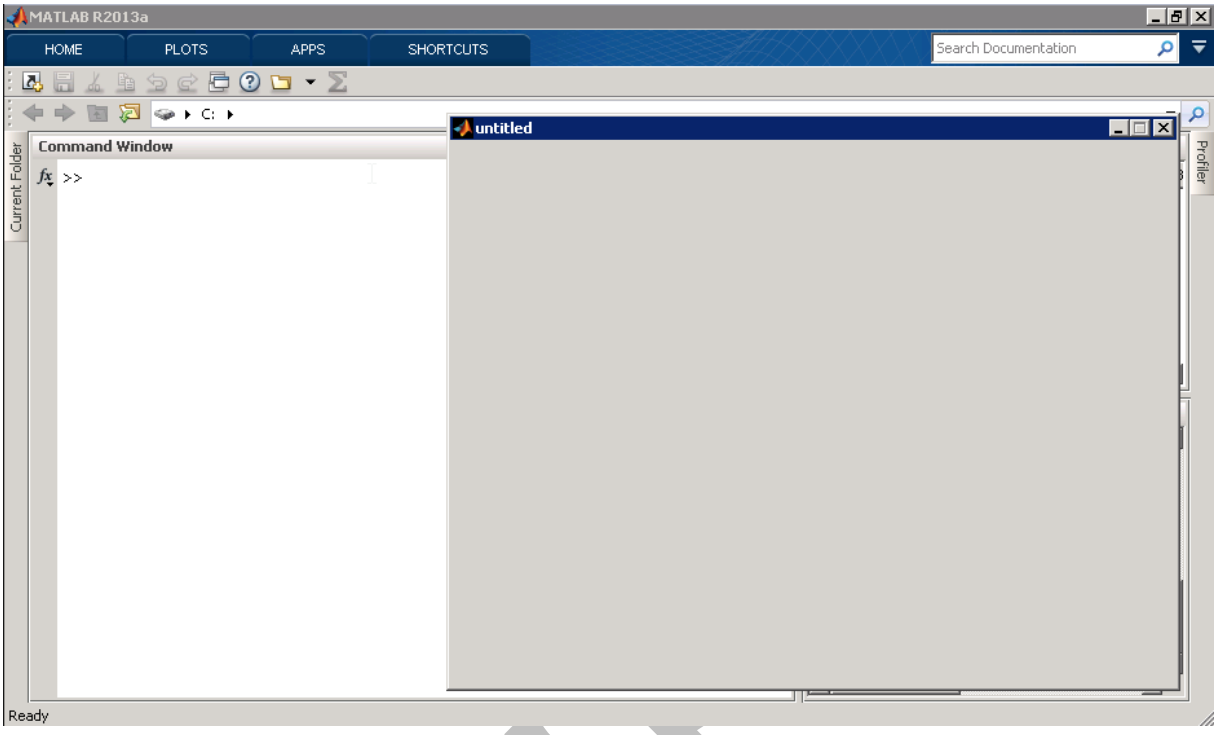
First thing to know is how to start the guide builder by typing **guide** in the *Command Window*:



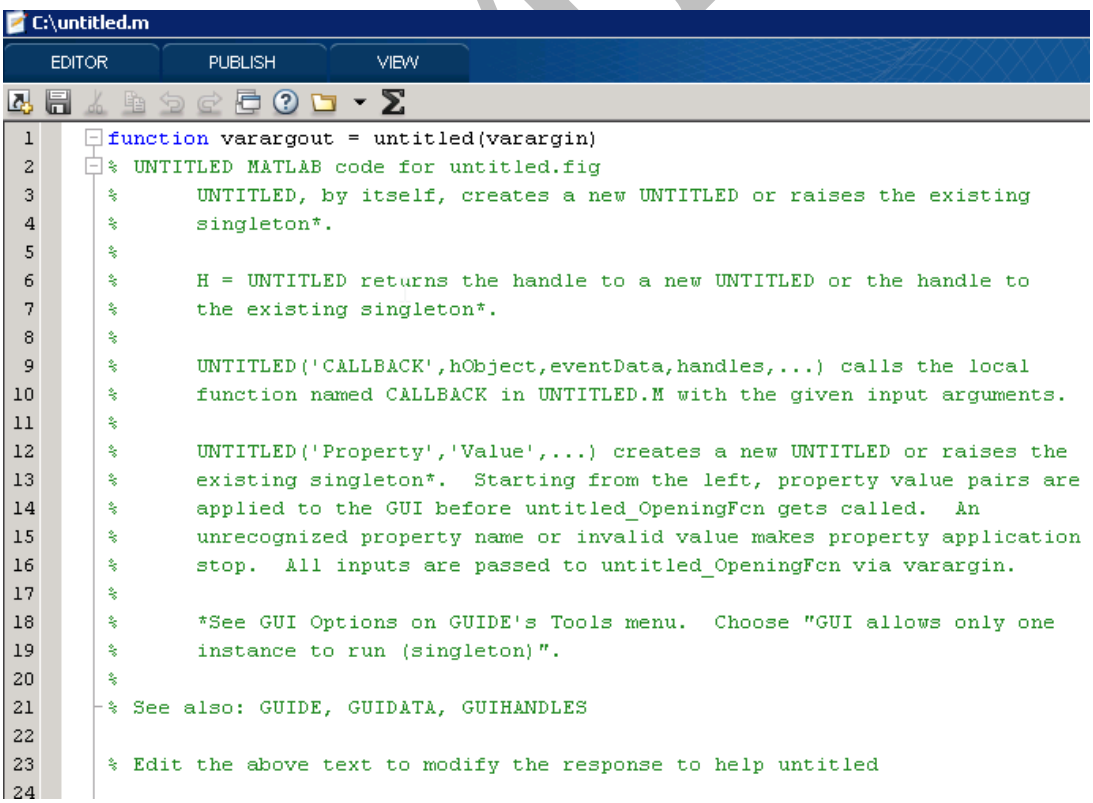
When you save you get two files:



When MATLAB is closed and you double click on the first (*.fig) you simply get MATLAB that open with the GUI:



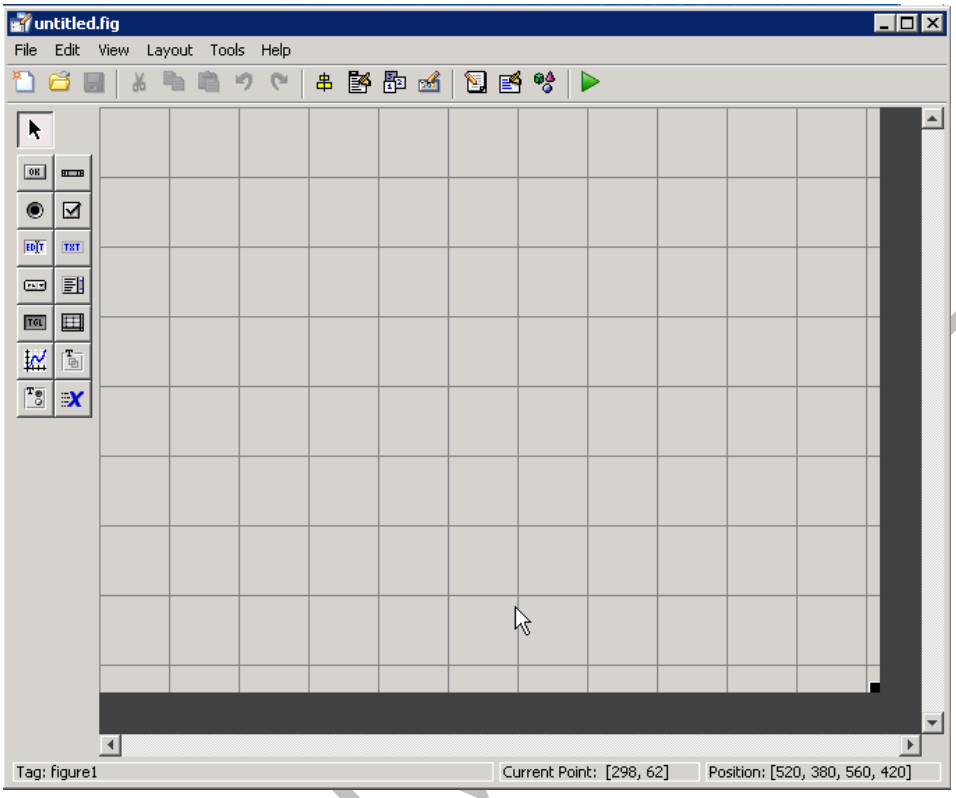
If you double click on the *.m file you get (this is only a part of the screenshot):



You can't run this file! If you remove the *.m file the GUI stills work because it is empty! But in reality never delete the *.m file associated to the figure because it will be used for callbacks!

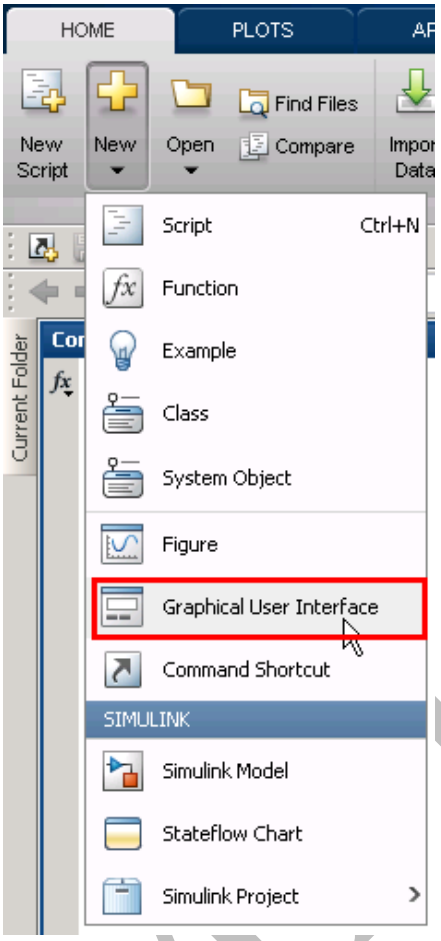
Also don't forget to make **Error Management**!!!

And we specify the figure file name in the field **Save figure as** and once validated, we get:



Another way to create a new GUI is to use the mouse:

SAMPLE



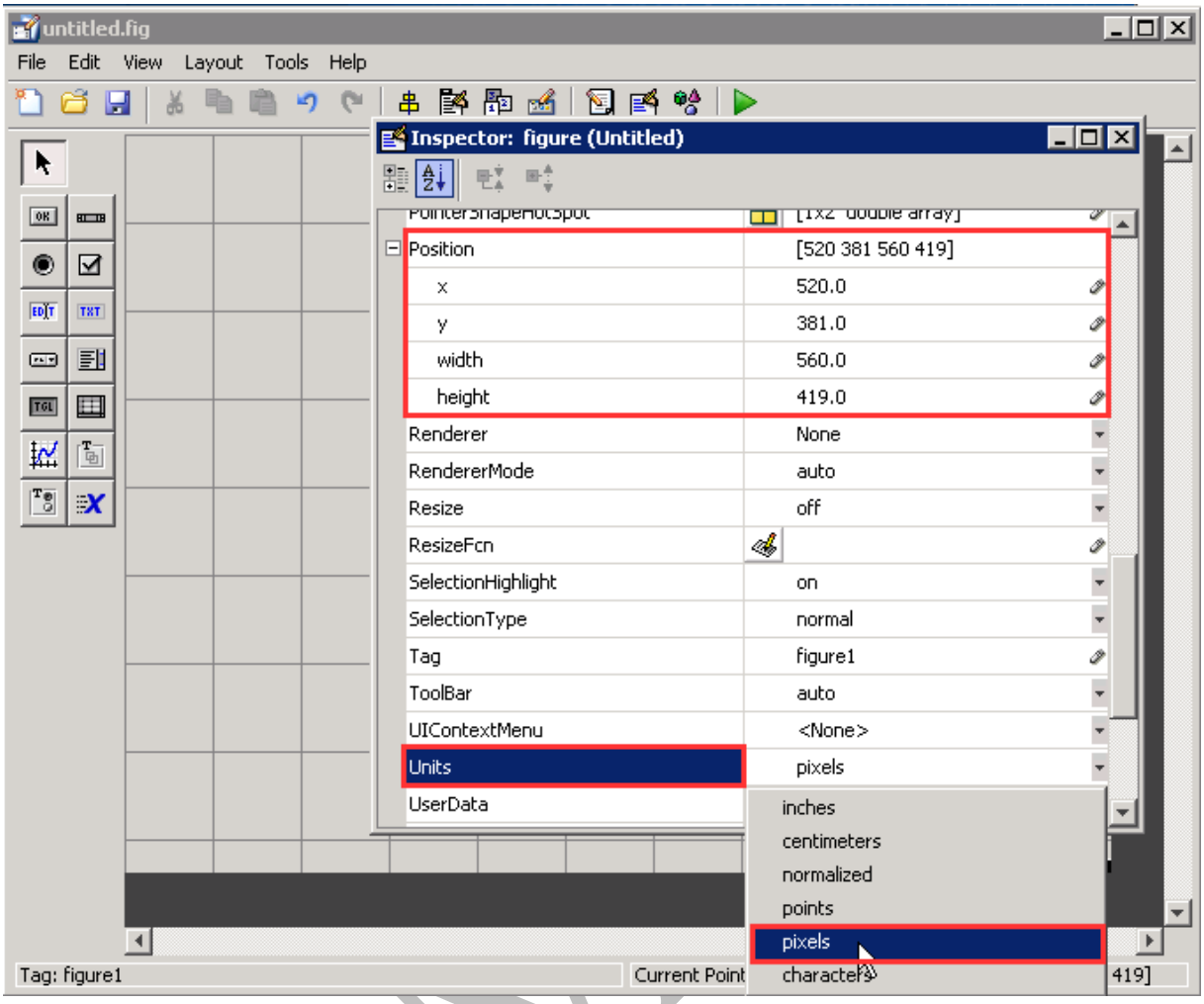
You can resize the userform by clicking on the lower-right corner and resize it.

Now to change the size and initial position with precision of a userform you can click on the userform **Inspector** button:

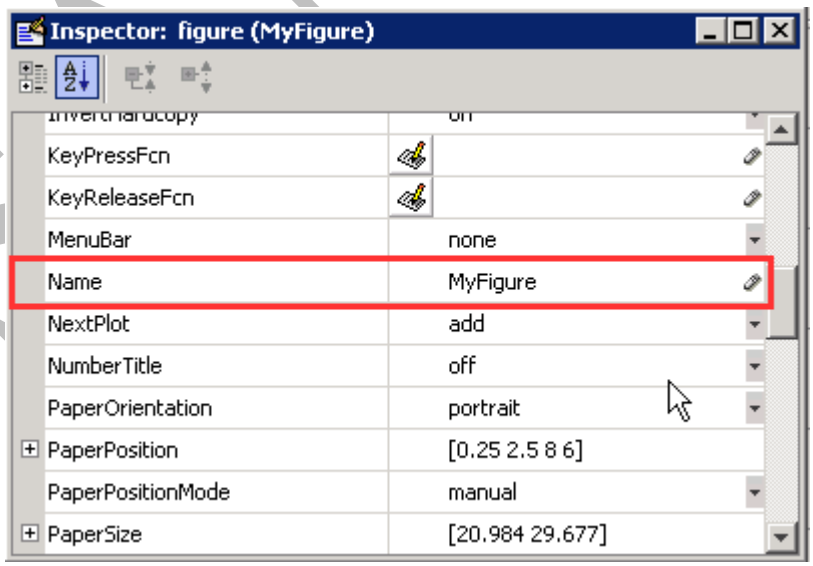


Then by scrolling down in the small window that will appear you will have the possibility to choose the measurement **Units** and also the **Position** properties:

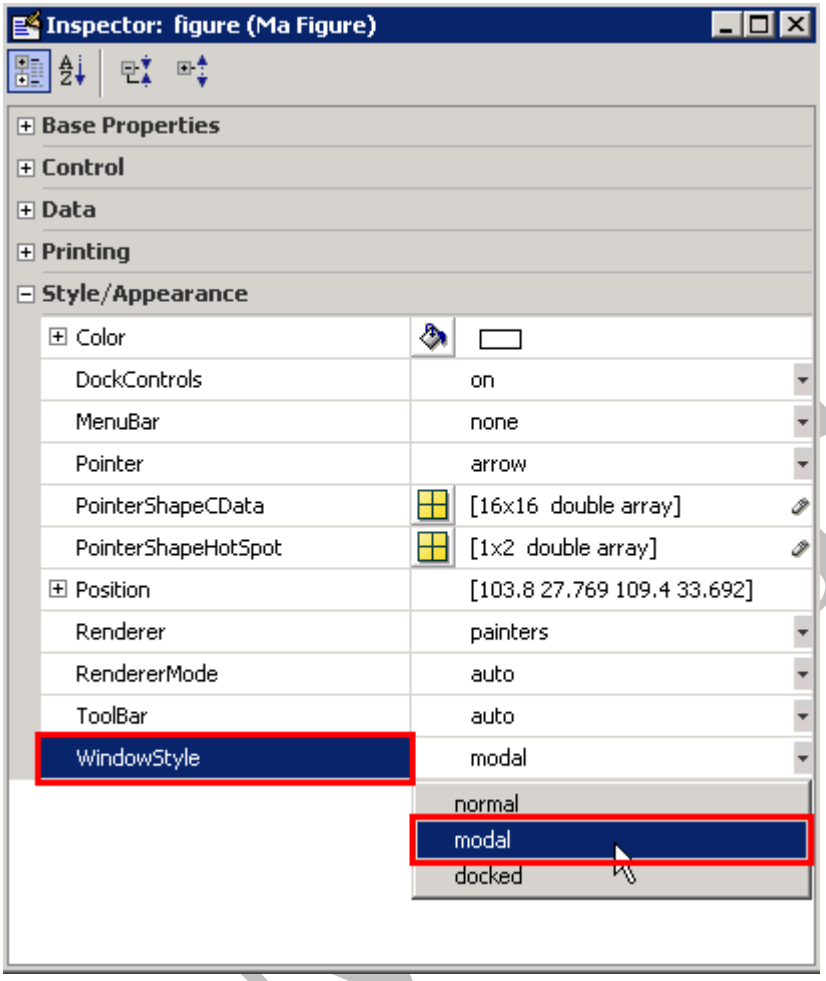
SAMPLE



To rename the GUI:

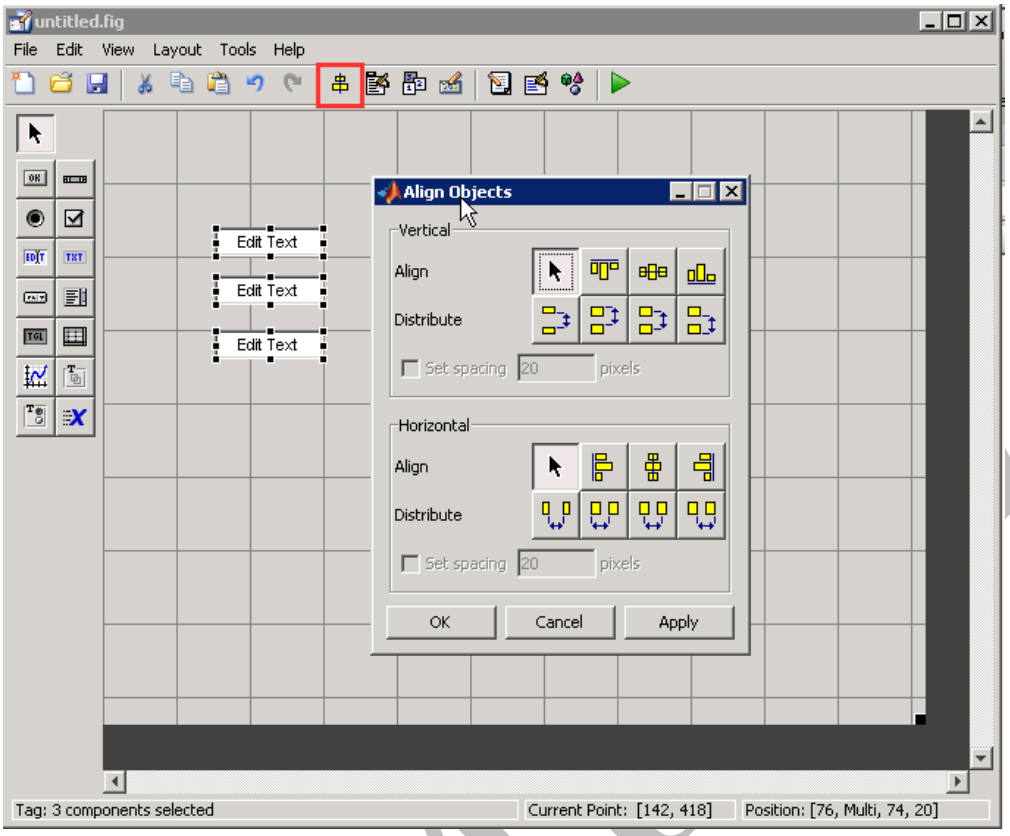


To make the GUI modal (so that the user cannot activate another window behind):

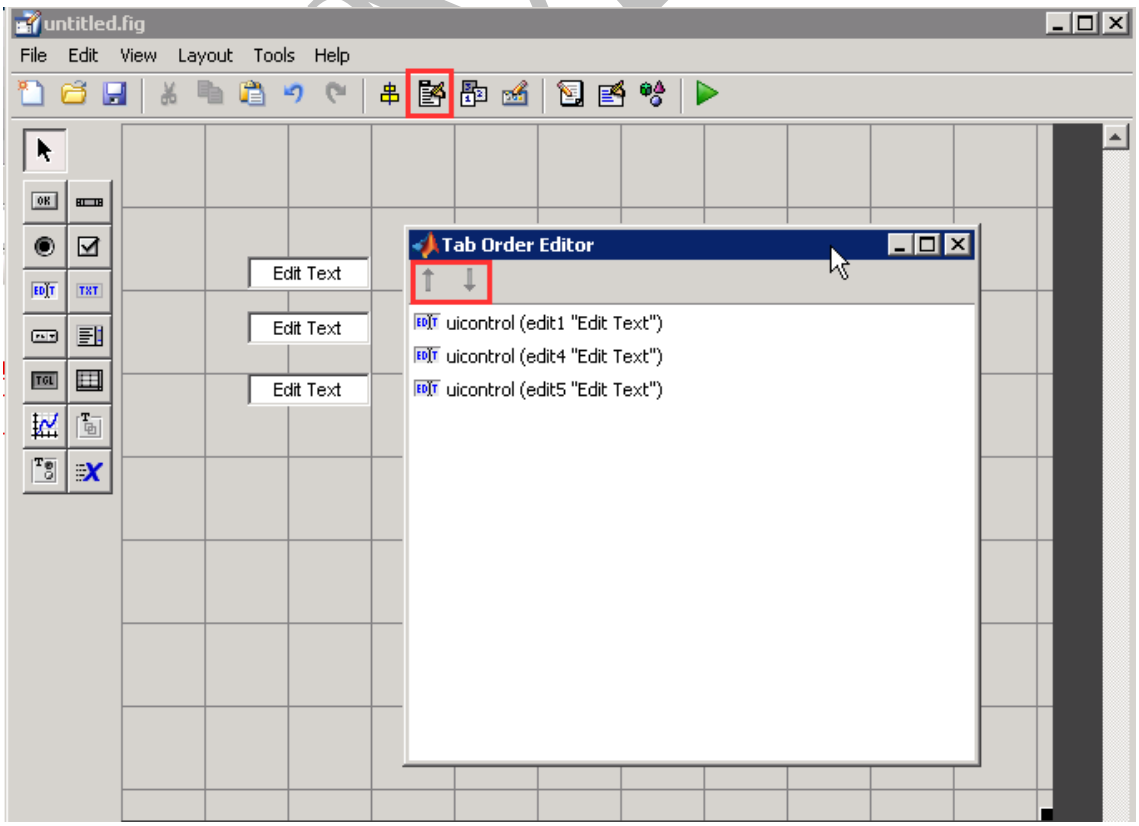


SAMPLE

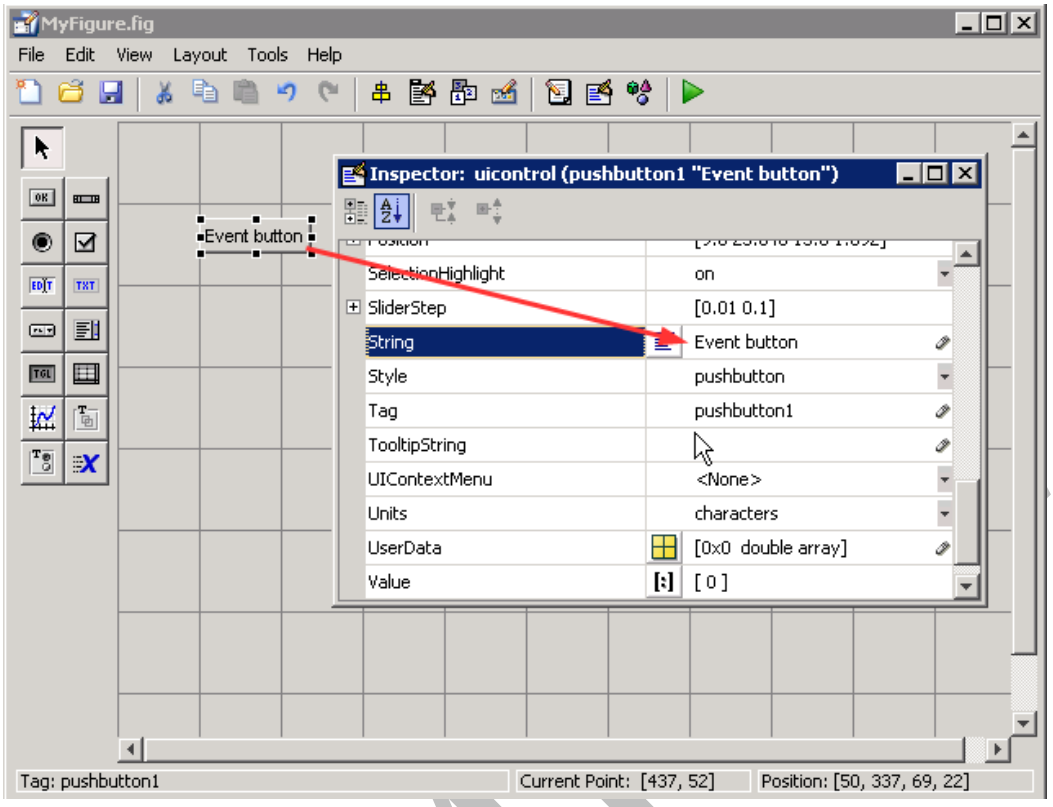
To align objects:



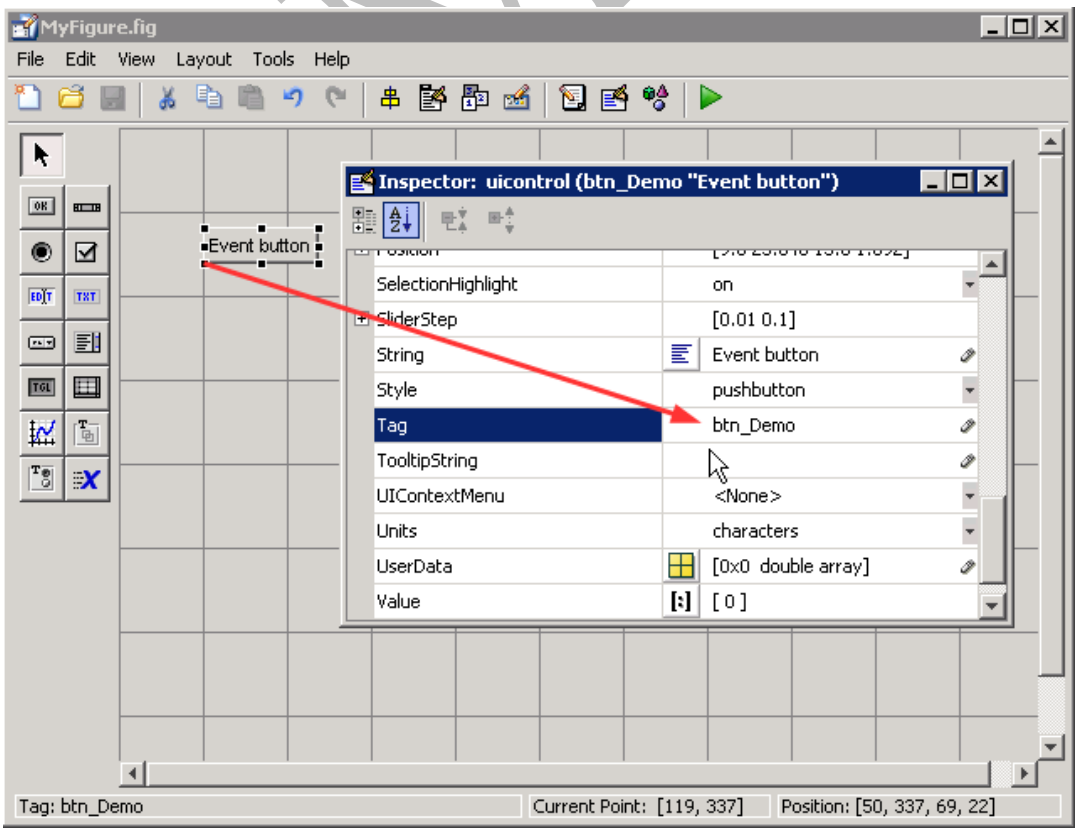
To choose tab order:



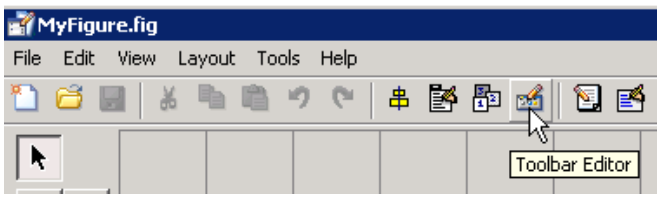
Change the legend name of a button:



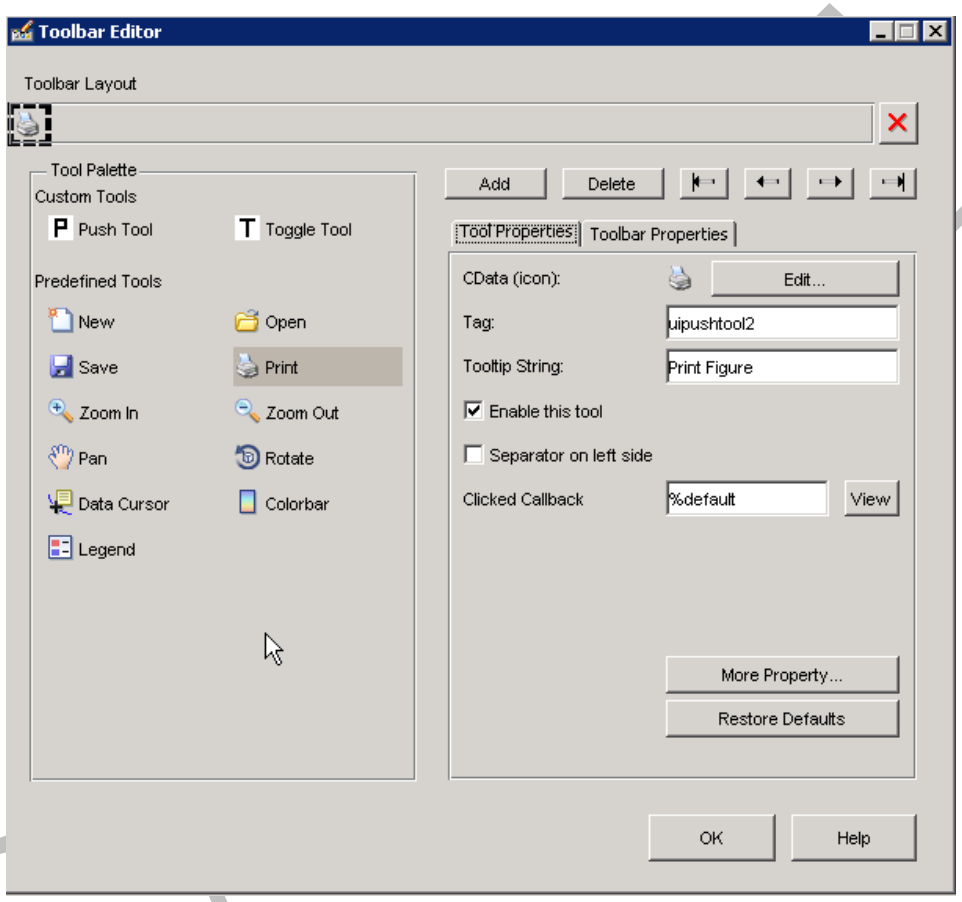
Change the technical name of the button (name also used for the event code) respecting Leszinky-Reddick nomenclature if possible:



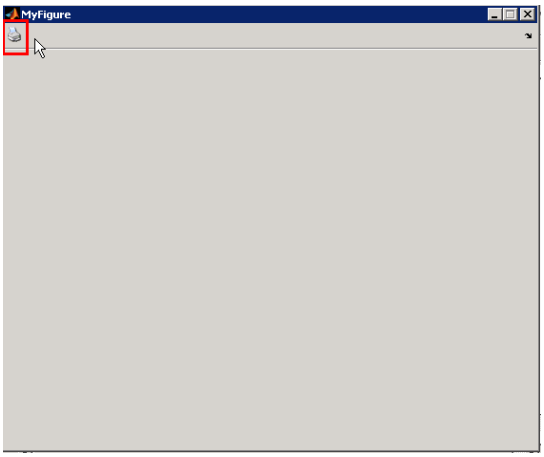
If you want to create toolbars on you figure this is easy. Just click on the **Toolbar Editor** Button:



And just drag and drop to buttons you want in the toolbar:

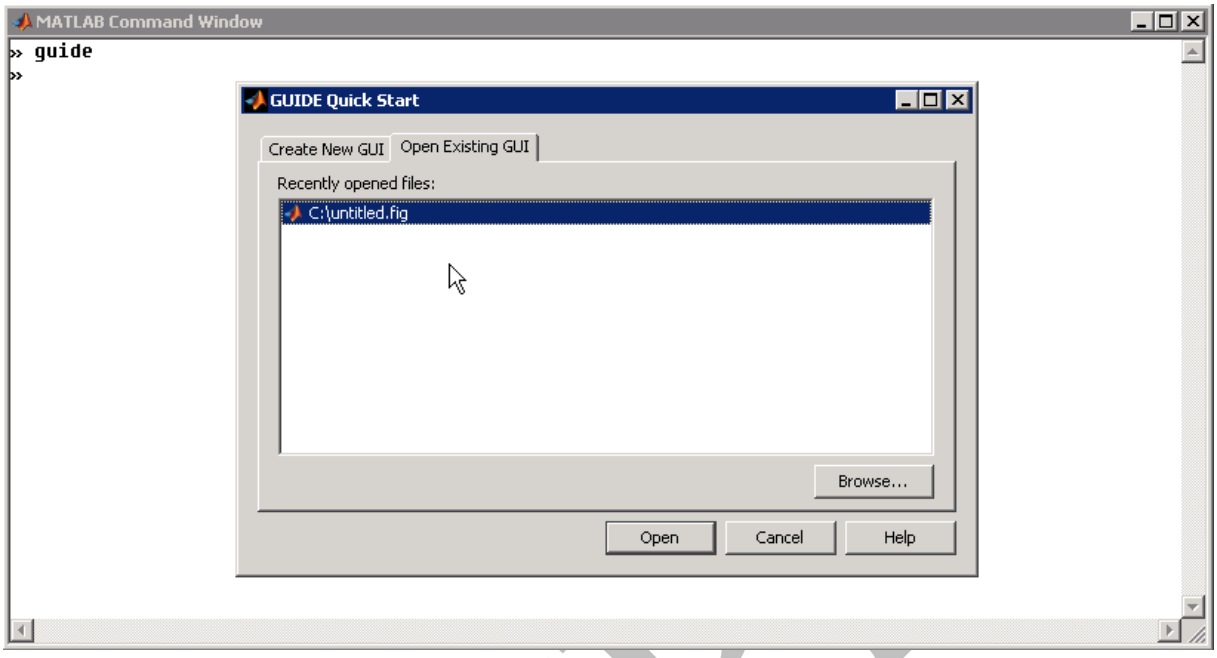


As you can see, it works well:



Edit existing GUI

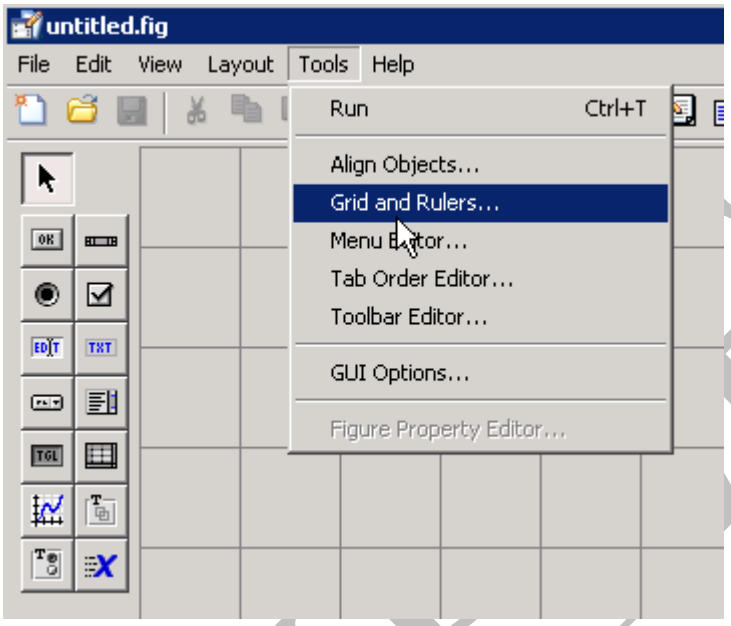
To edit an existing GUI we use the same technic as when we create one:



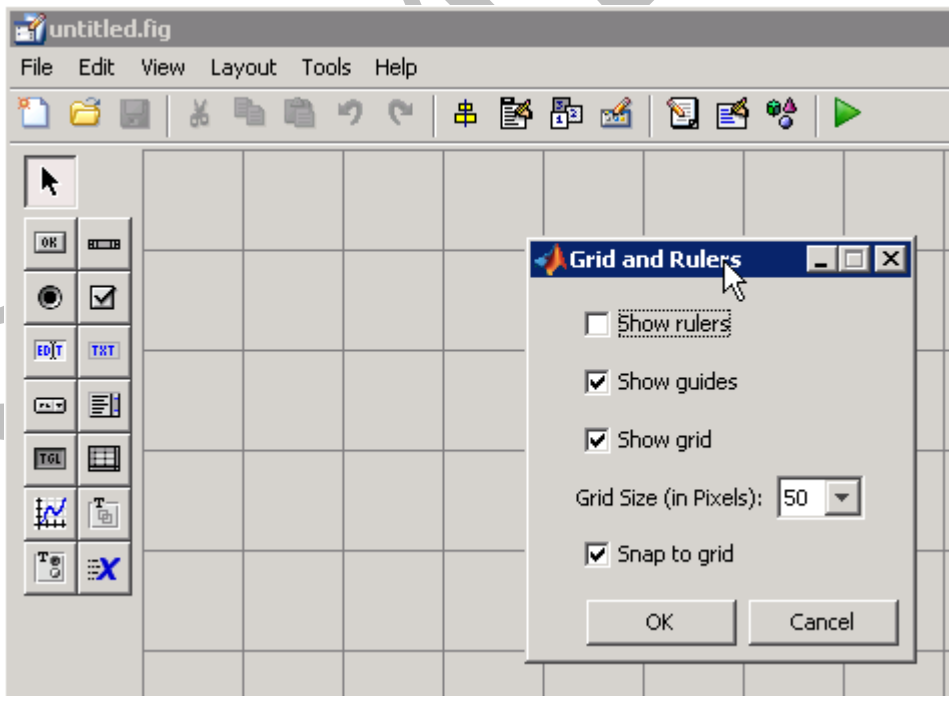
SAMPLE

Manage static texts (legends)

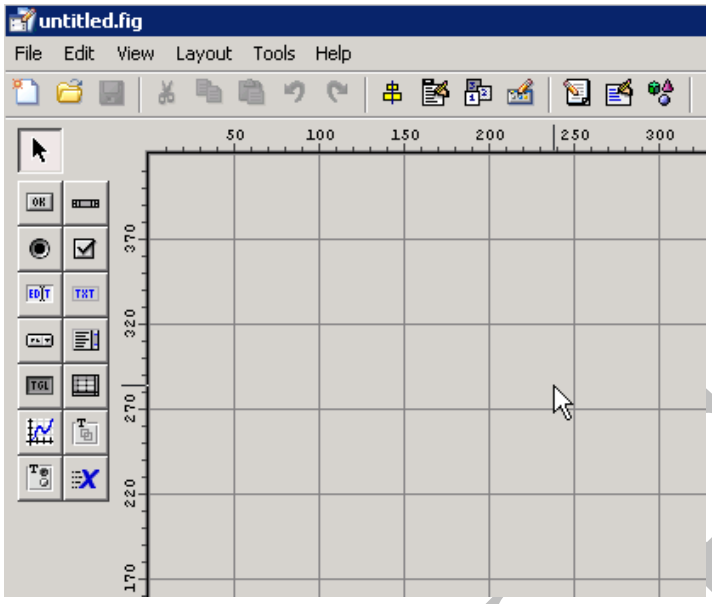
Add **Static Texts**  components is very easy you just need to draw them like on any Microsoft Office software. To quickly align them you can use the Grid and Rulers options:



So you get:



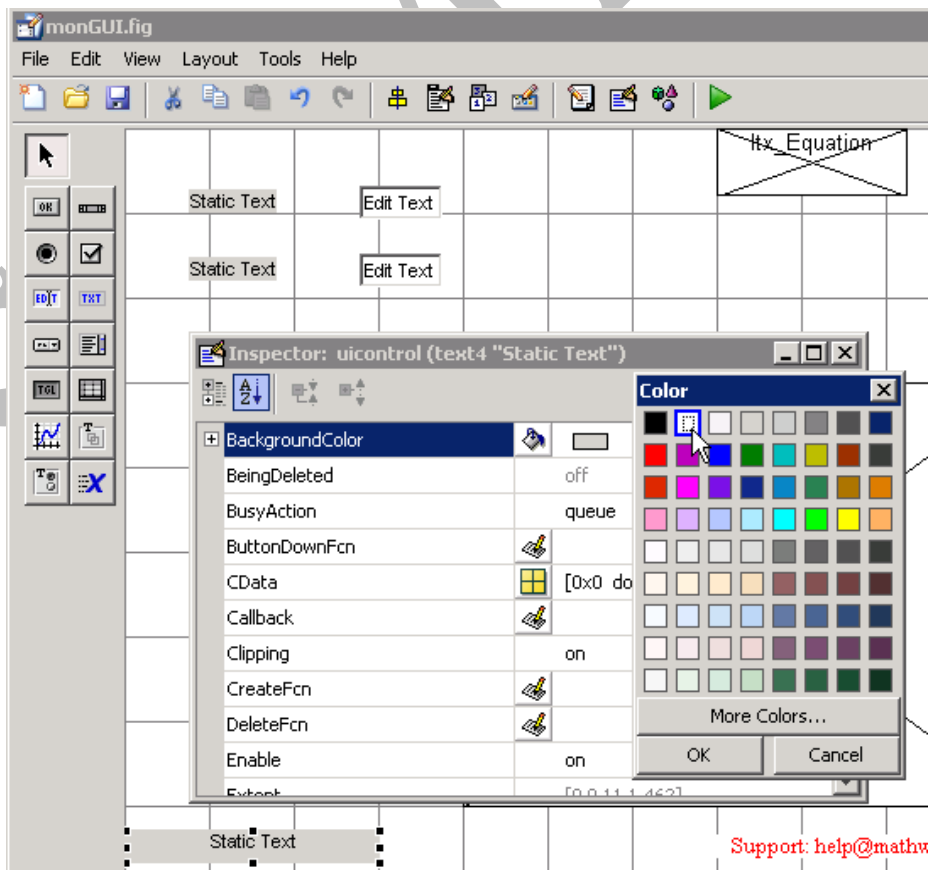
Activate **Show rulers** and the final result will be:



By dragging the rulers you can crate guides that will greatly help you to align objects proactively.

Once you have added some **Static Text** and **Edit Text** components, let us have a look now to the most used properties.

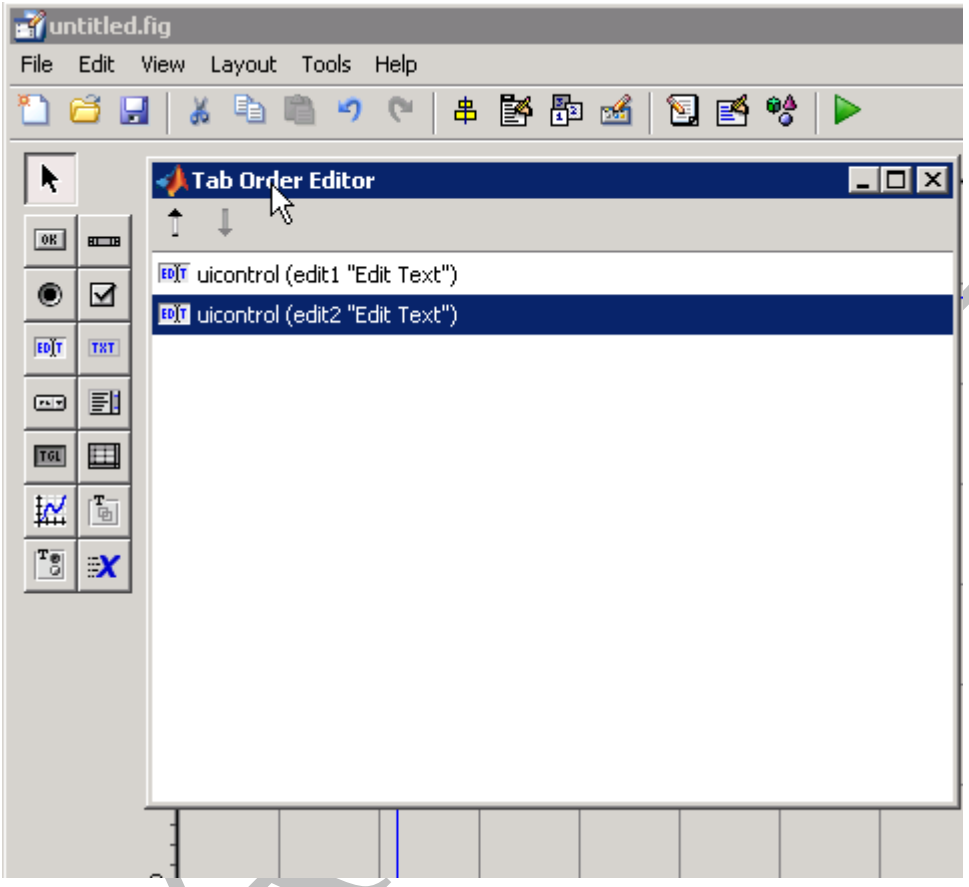
First we change the background color with the property **BackgroundColor**:



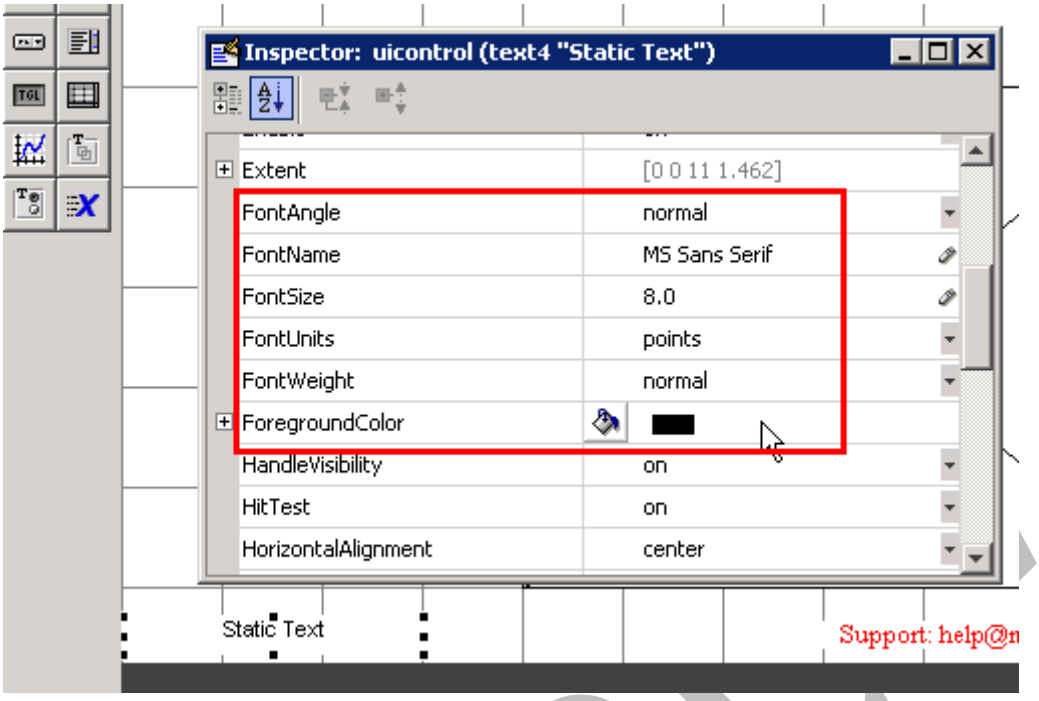
Another important stuff for **Edit Text**  components is to define Tabulation order:



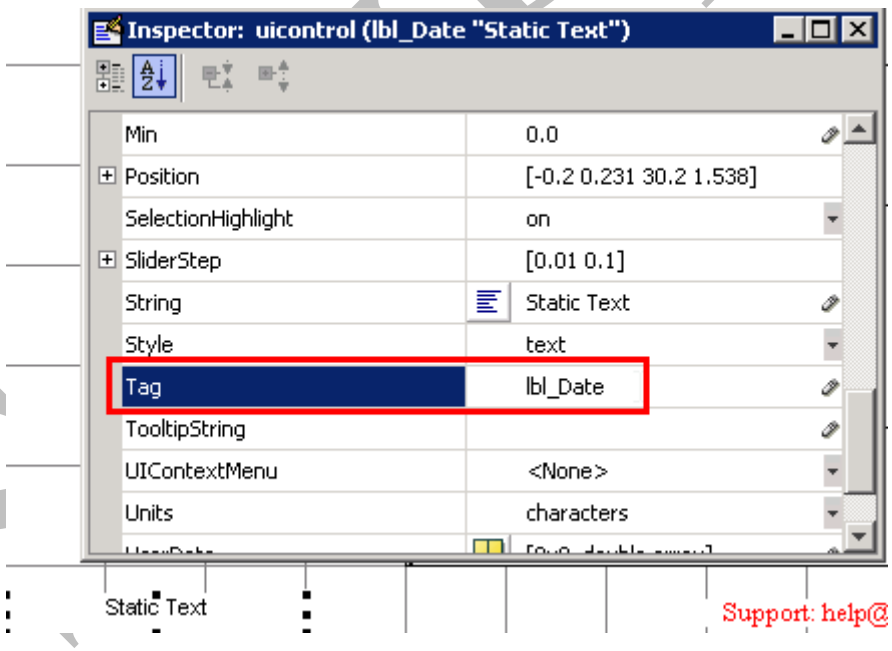
So you get:



You can also change classical font properties but you have to see that you cannot mix in the same **Static Text** component different text styles:



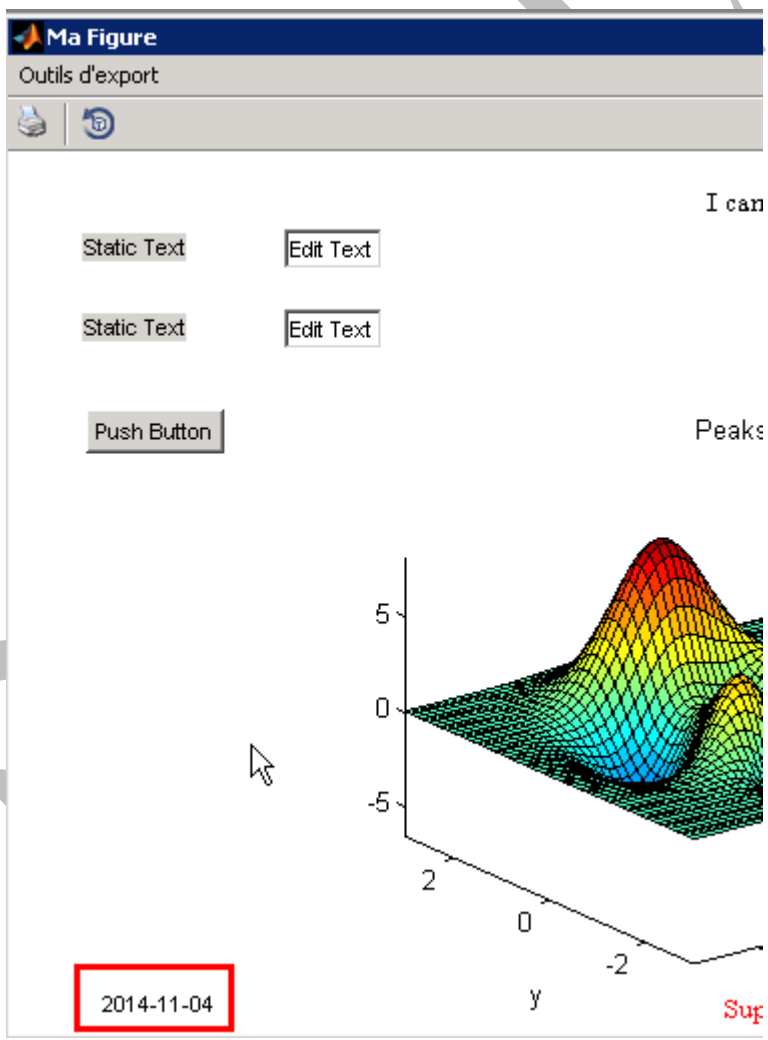
As always, we change the component **Tag** property respecting Leszinky-Reddick nomenclature:



Now we go in the script editor associate to the GUI and the *OpeningFcn* event (callback) we add the following:

```
monGUI.m
47    % --- Executes just before monGUI is made visible.
48    function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.%...%
54
55    % Choose default command line output for monGUI
56    handles.output = hObject;
57
58    % Update handles structure
59    guidata(hObject, handles);
60
61    set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
```

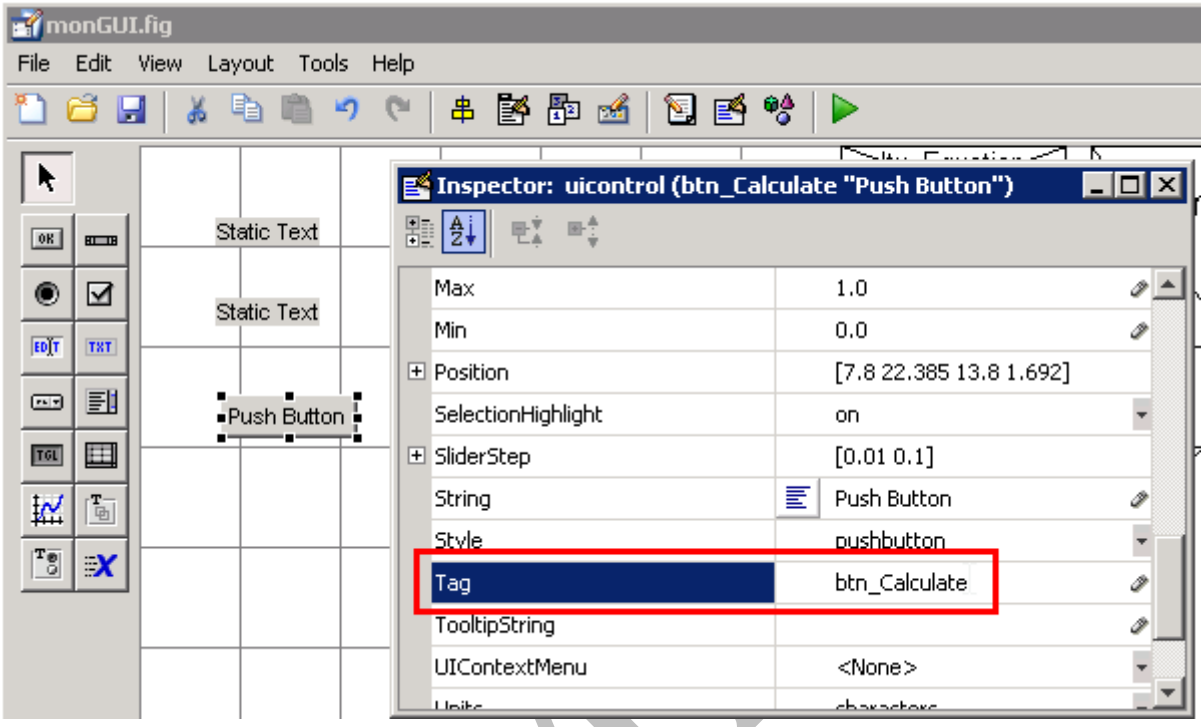
Once this done, we you or ur user open the GUI they get:



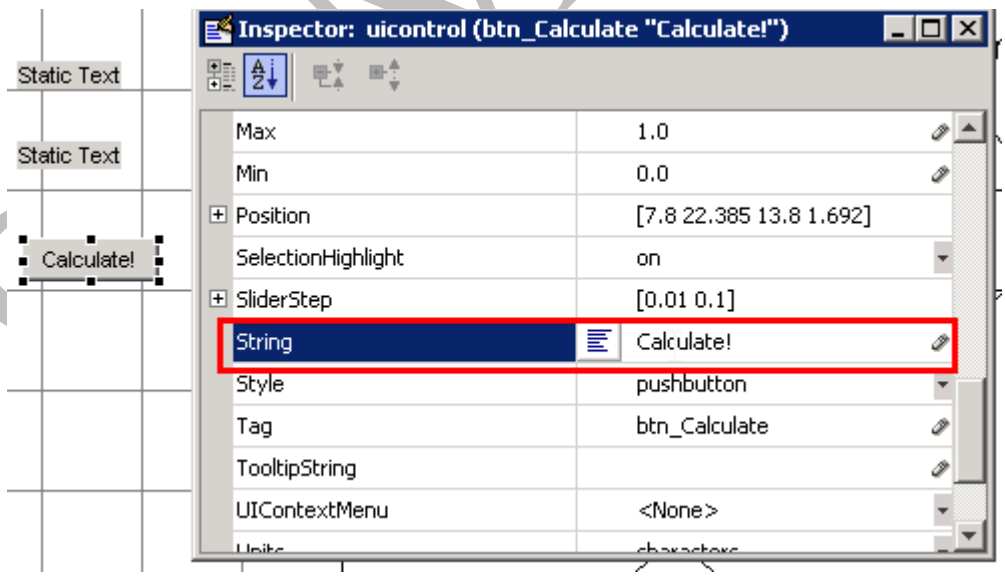
Manage buttons and associate a callback

Adding a button on a GUI is very easy. In fact what interest us is to associate a callback to it.

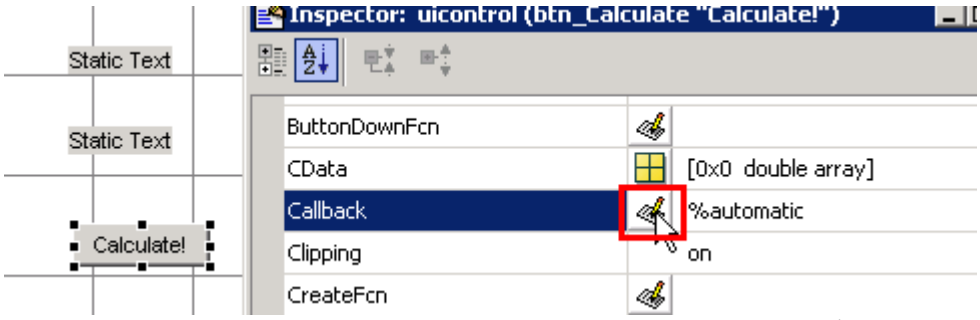
First rename as always the button **Tag** property following Leszinky-Reddick nomenclature:



Change also the string in the button to be more explicit...:



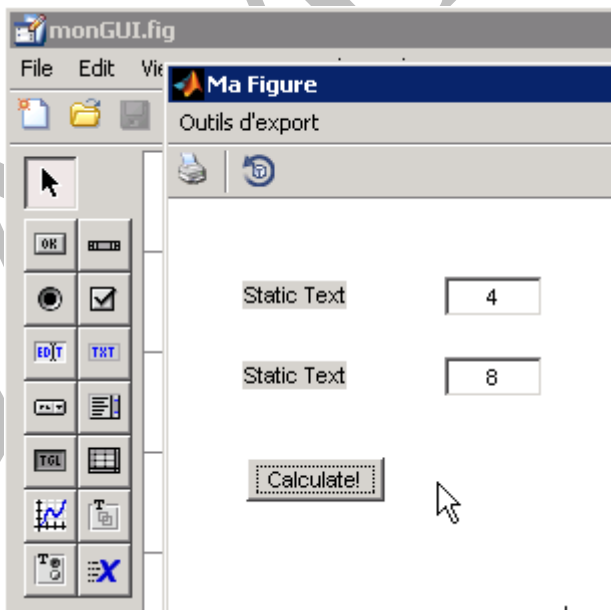
And after we change its associated **Callback** (event) property:



And we add and simple multiplication where we read the value from the first **Edit** field (*edit1*) and we write the result in the second one (*edit2*):

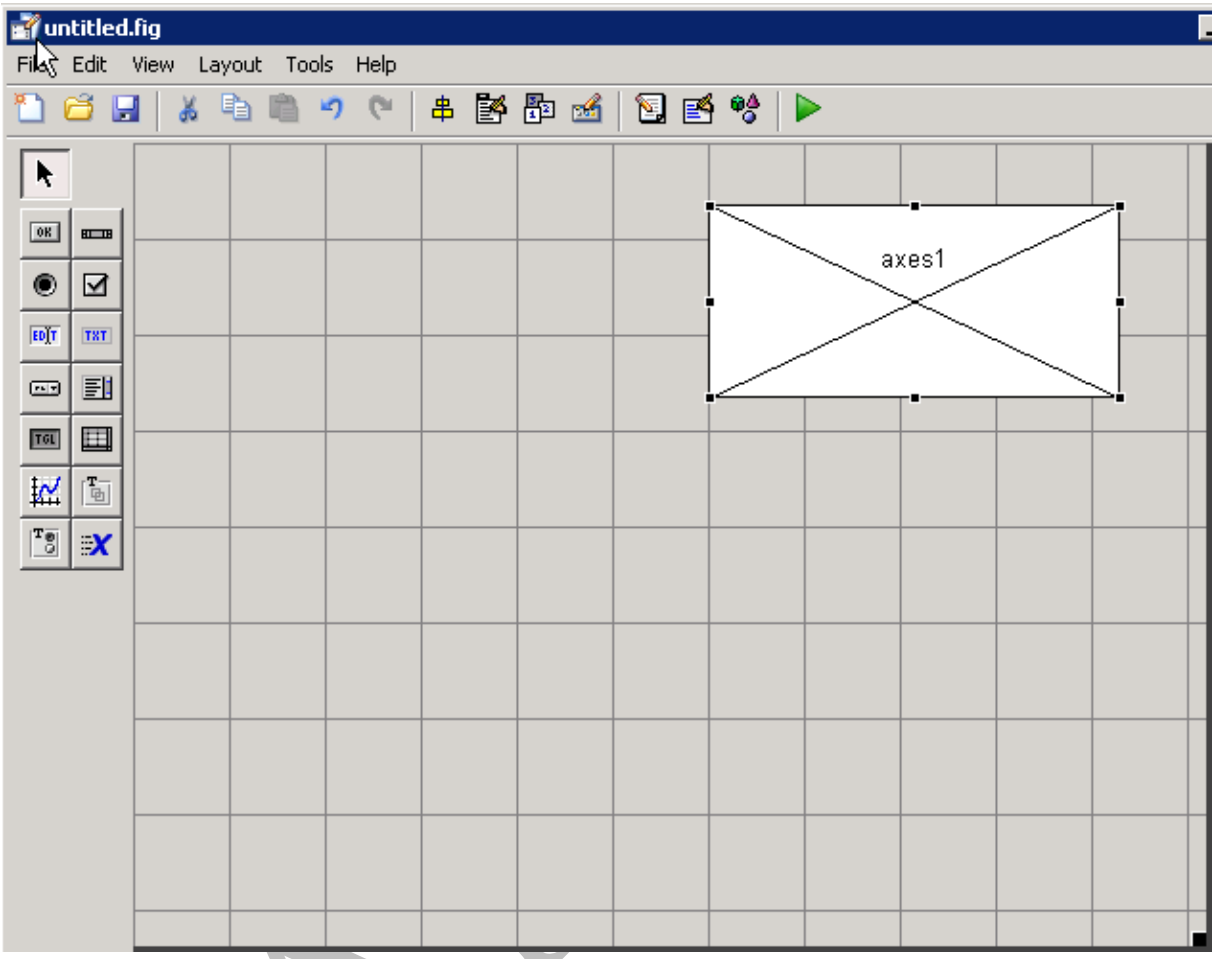
```
148
149     % --- Executes on button press in btn_Calculate.
150     function btn_Calculate_Callback(hObject, eventdata, handles)
151     % hObject    handle to btn_Calculate (see GCBO)
152     % eventdata  reserved - to be defined in a future version of MATLAB
153     % handles    structure with handles and user data (see GUIDATA)
154
155 -   sng_Value=str2double(get(handles.edit1,'String'))*2;
156 -   set(handles.edit2,'String',sng_Value);
```

So the results will looks like:

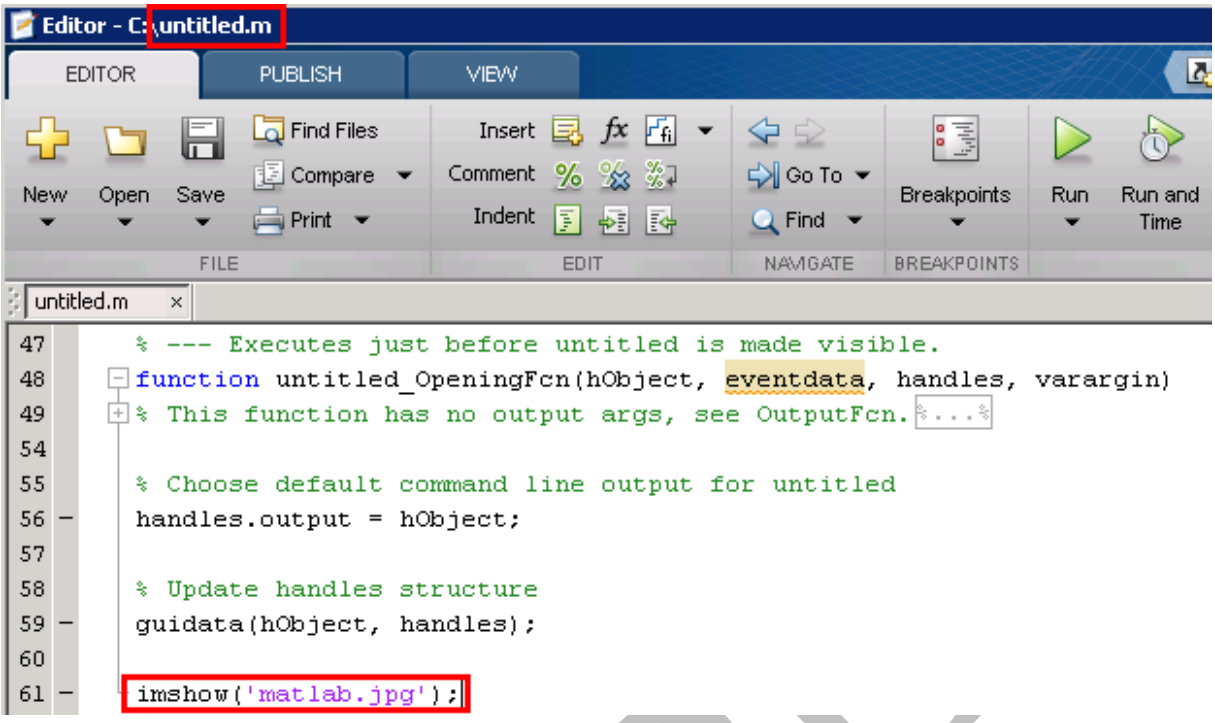


Create axes area for figures, images,

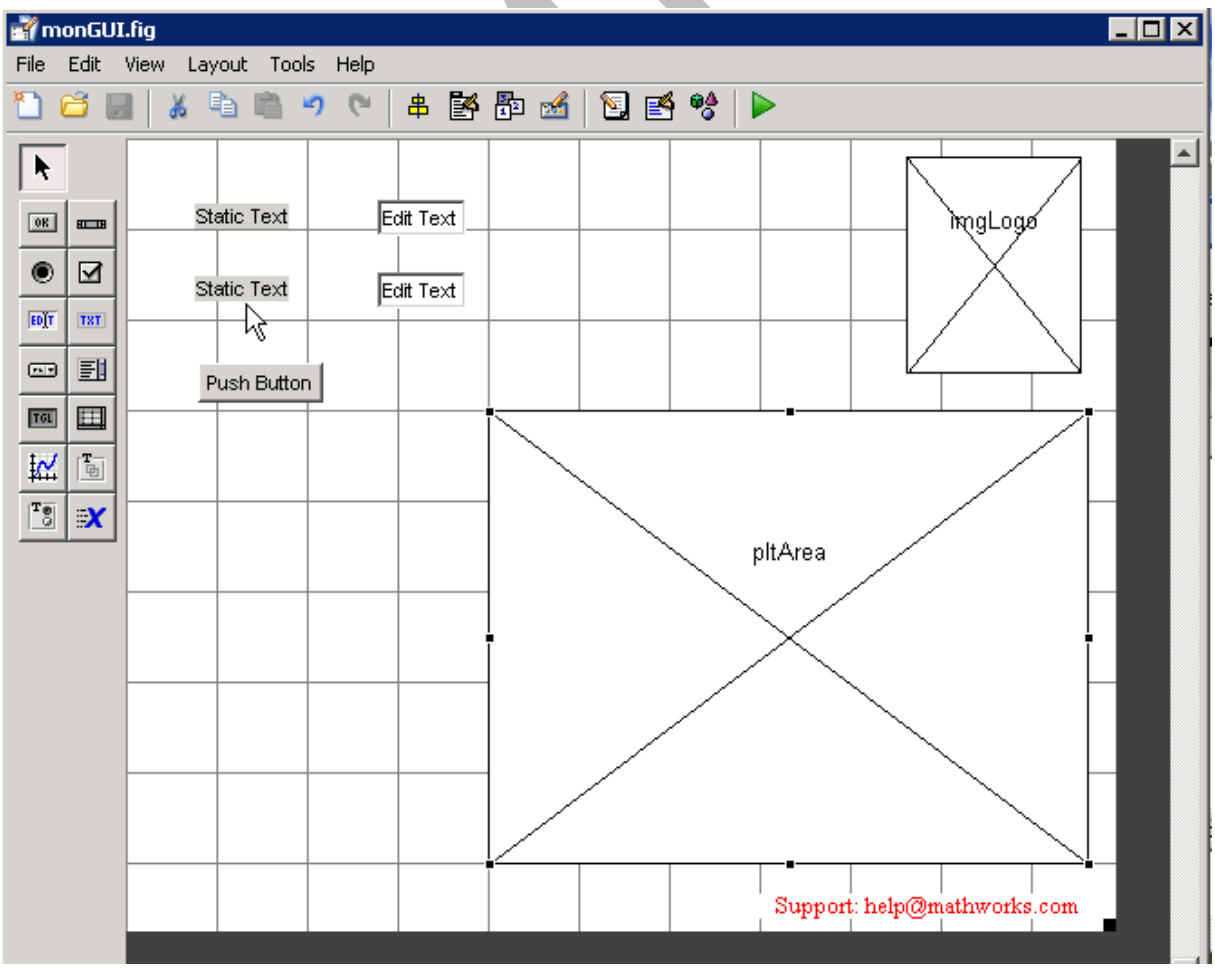
You have for this subject to consider two scenarios. The first is that you only have one unique axes object on your userform like here:



Then to fill it with an image it's quite easy. You just go in the associated script file and add in the *OpeningFcn* event the following line:



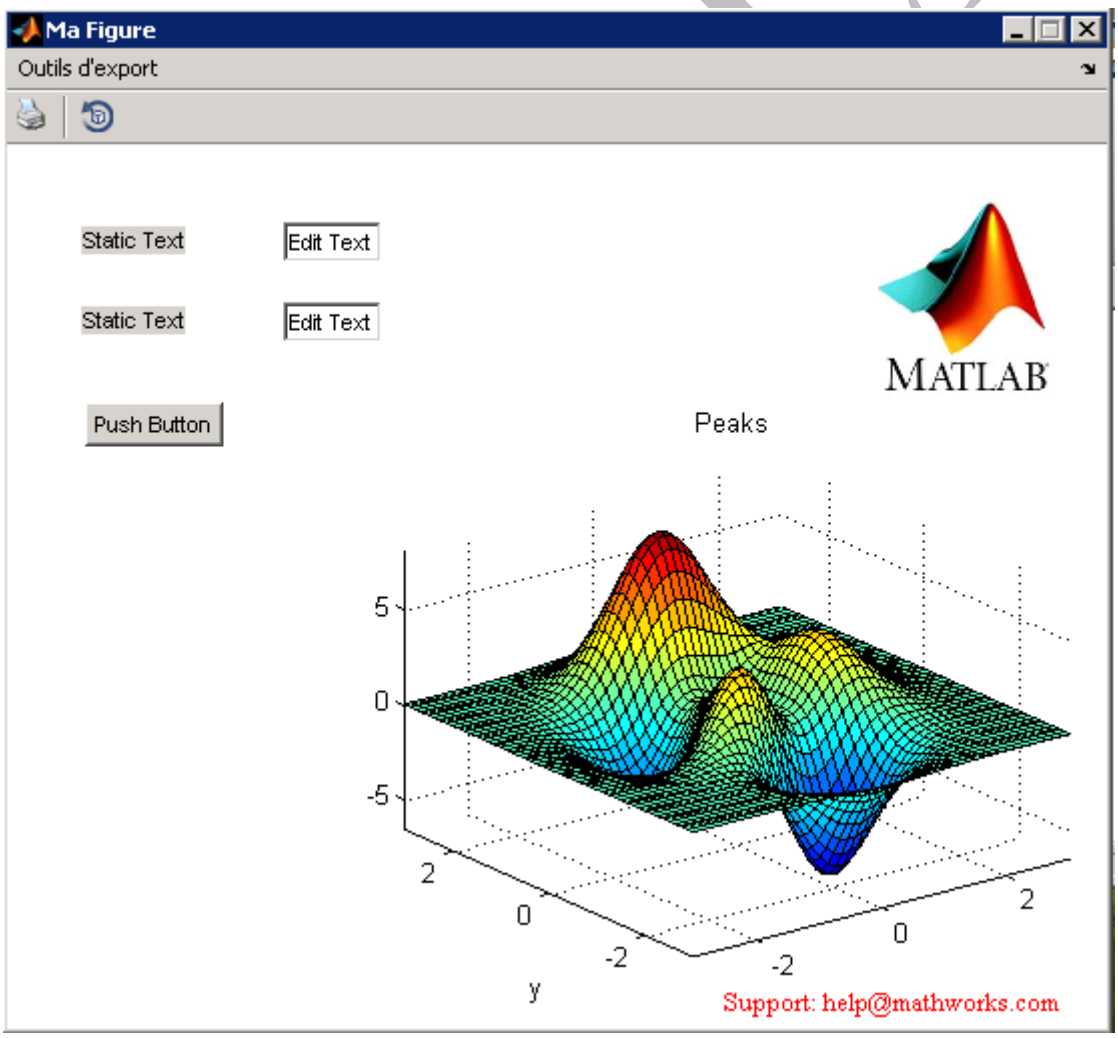
When there are more than one axes area it is not much more difficult. Consider first the following situation where we change the **Tag** property of the two axes components:



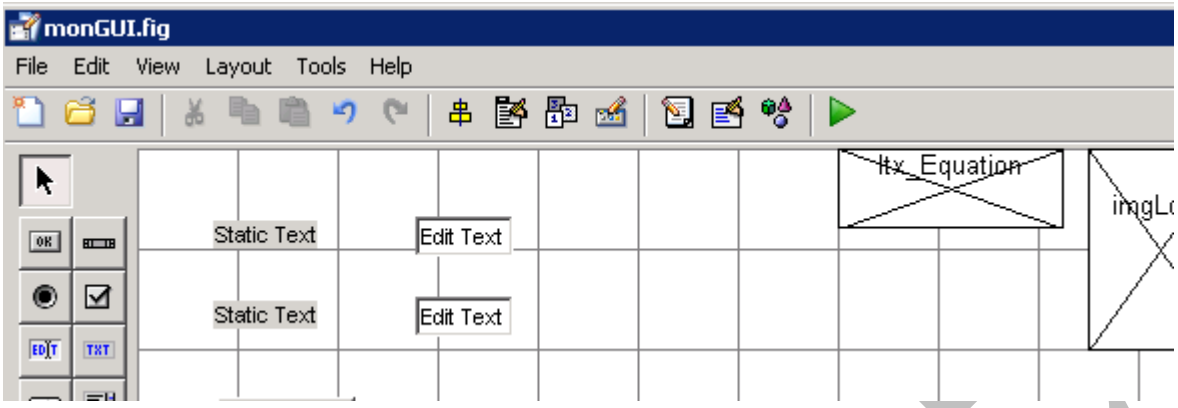
We just change the script as following:

```
monGUI.m
48  function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49  % This function has no output args, see OutputFcn.
54
55  % Choose default command line output for monGUI
56  handles.output = hObject;
57
58  % Update handles structure
59  guidata(hObject, handles);
60  axes(handles.imgLogo);
61  imshow('matlab.jpg');
62  axes(handles.pltArea);
63  peaks;
64
```

And we get:



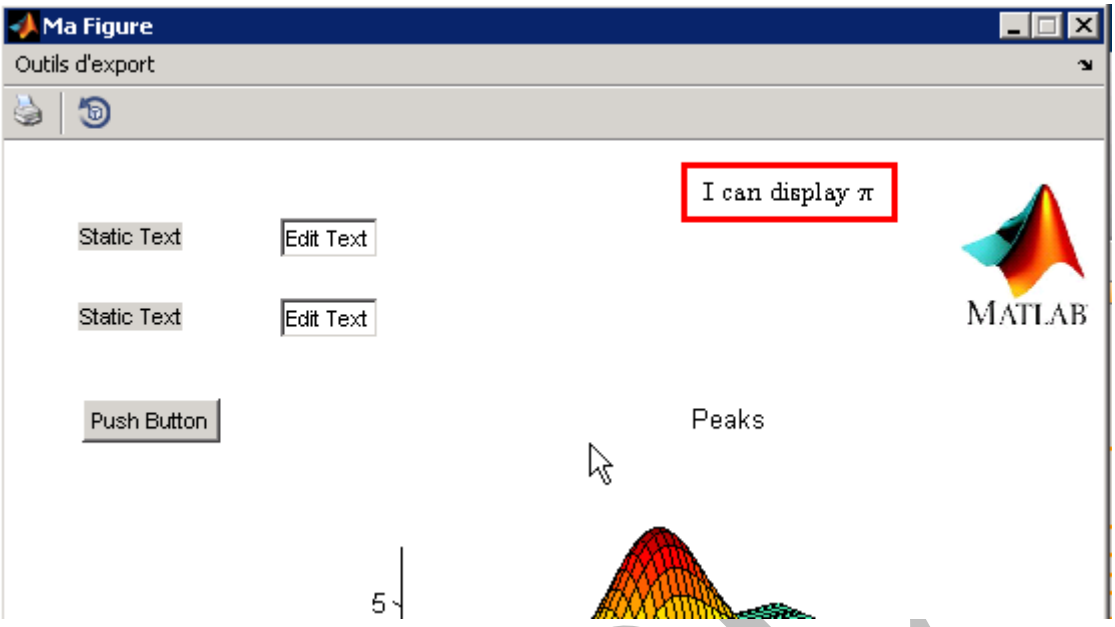
The use of axes seems also the only way actually to write LaTeX on a GUI. First add an axes component where you want and resize it and change its **Tag** property (here we choosed *ltx_Equation*):



One this is done, the idea in this particular example will be to write a LaTeX text once the GUI is opened. For this purpose we go in the script editor associated to the GUI and we write in the *OpeningFcn* event (callback) the following:

```
monGUI.m
47    % --- Executes just before monGUI is made visible.
48    function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.
54
55    % Choose default command line output for monGUI
56    handles.output = hObject;
57
58    % Update handles structure
59    guidata(hObject, handles);
60
61    axes(handles.ltx_Equation);
62    axis off;
63    text(0,0.5,'I can display $\pi$','interpreter','latex',...
64    'horiz','left','vert','middle')
```

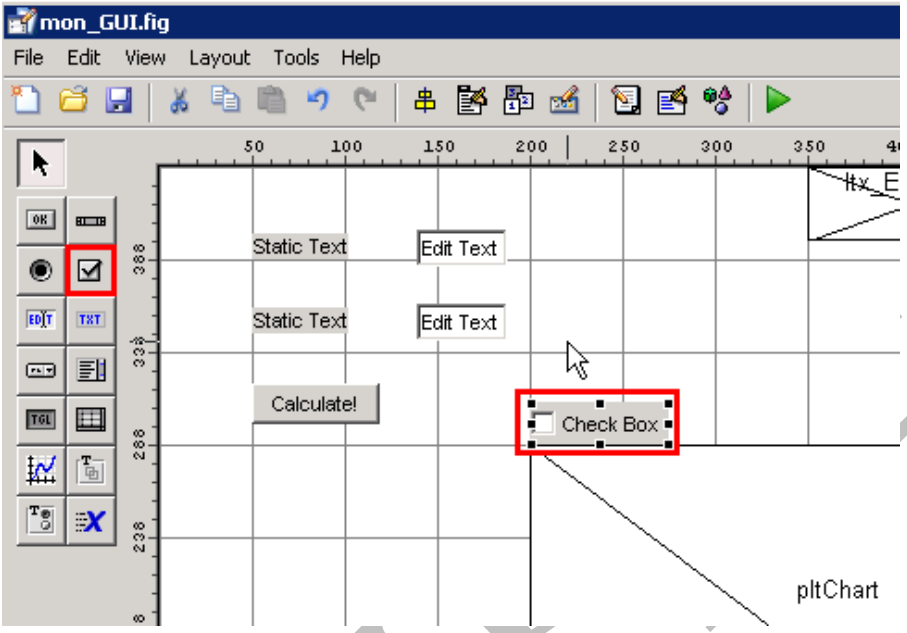
Once this is done, when we or the user opens the GUI, we get:



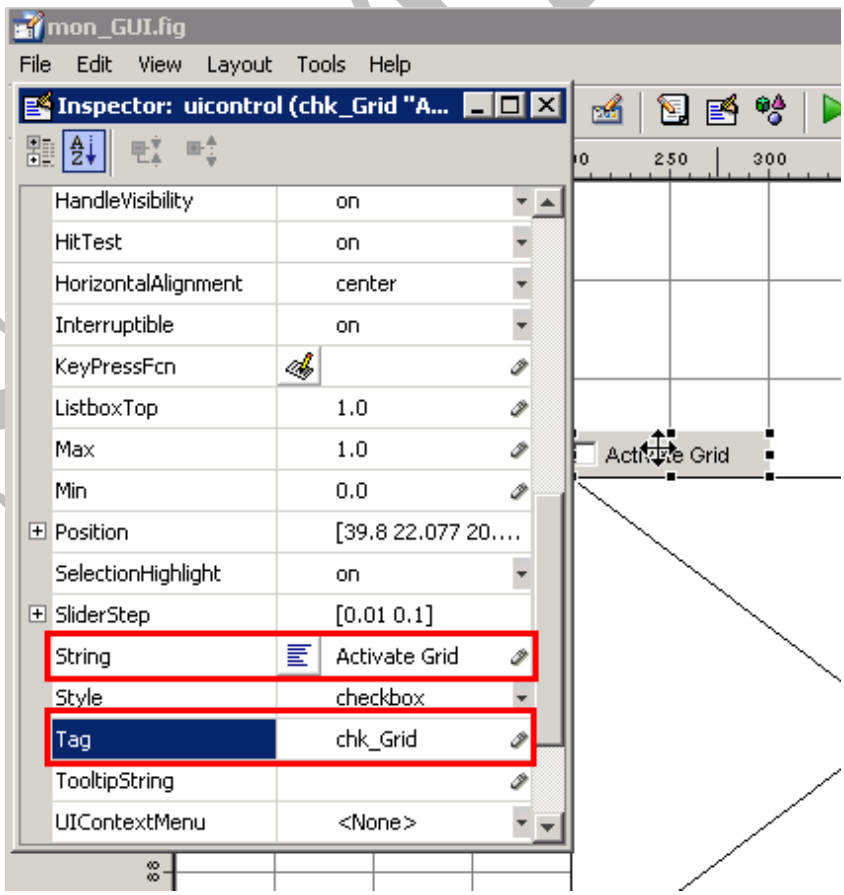
SAMPL

Manage check-boxes and associate a callback

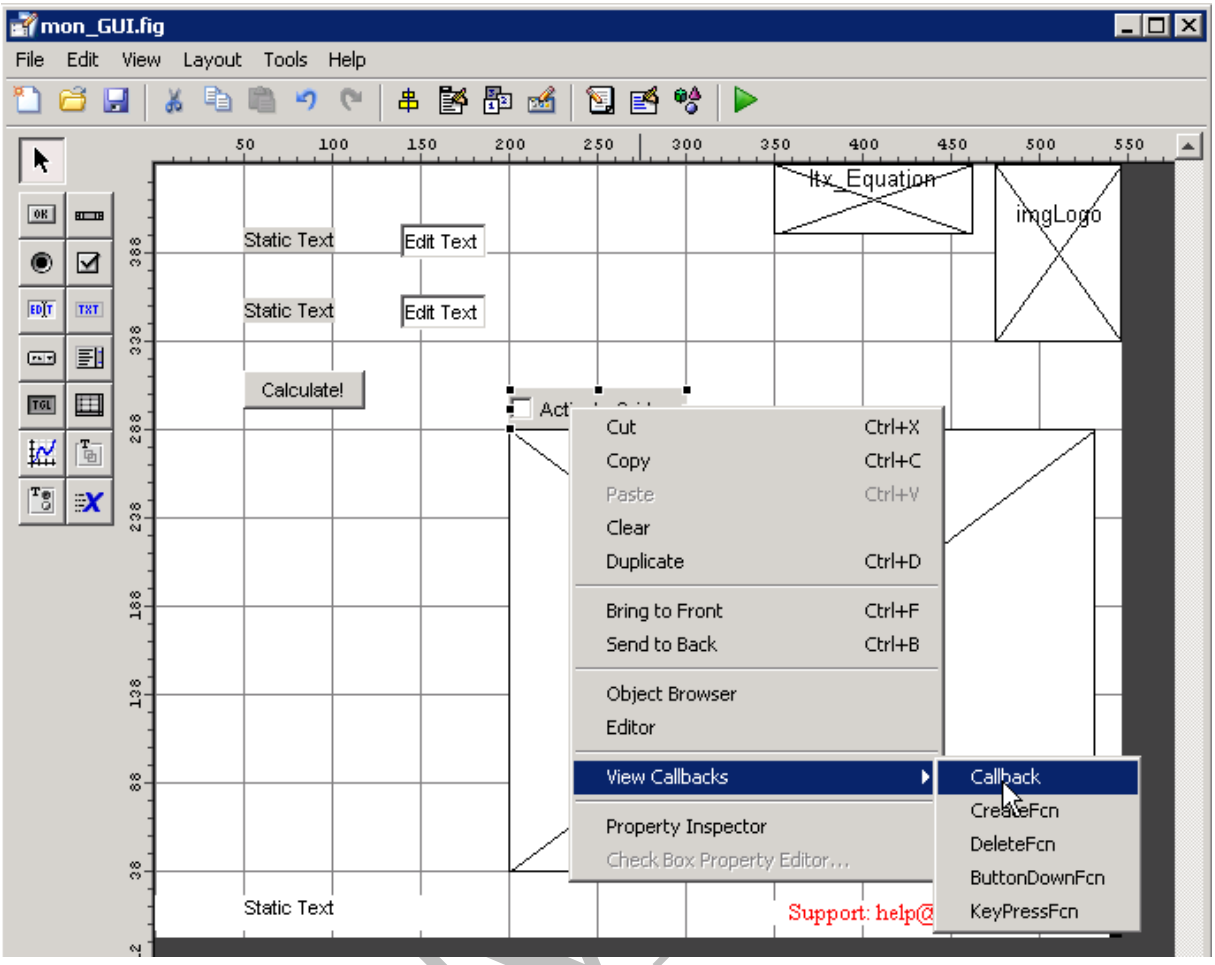
We will see the idea of using **Checkboxes** by a very simple case. First add such a component:



We change the classic stuff:



Now we go to the Callback of the checkbox:

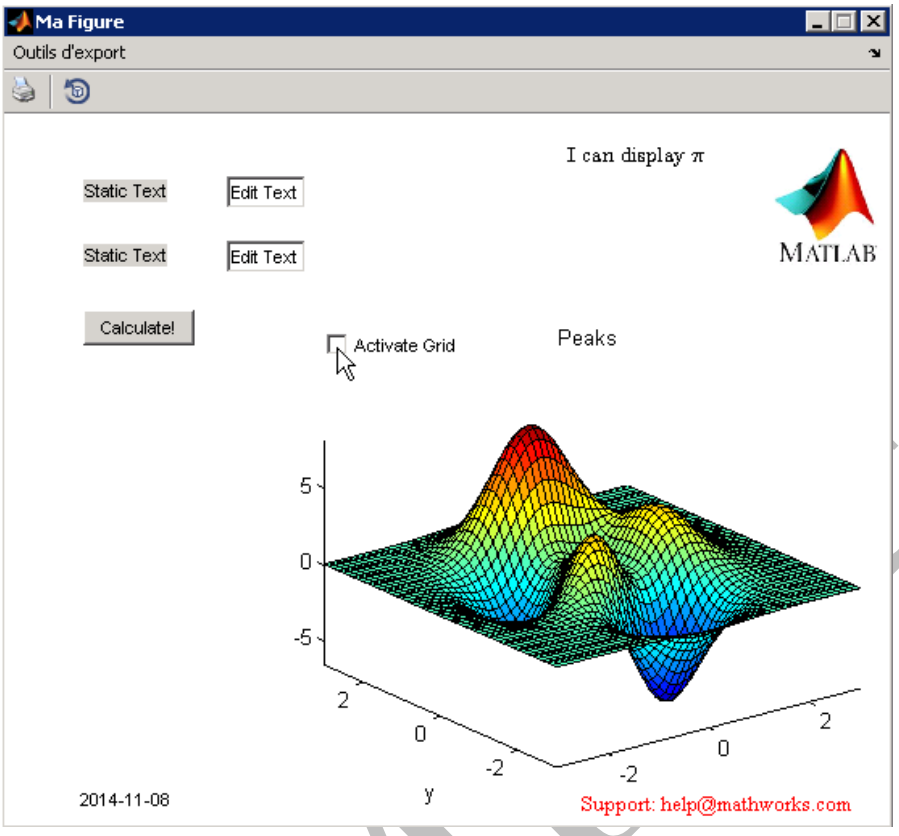


And so we write:

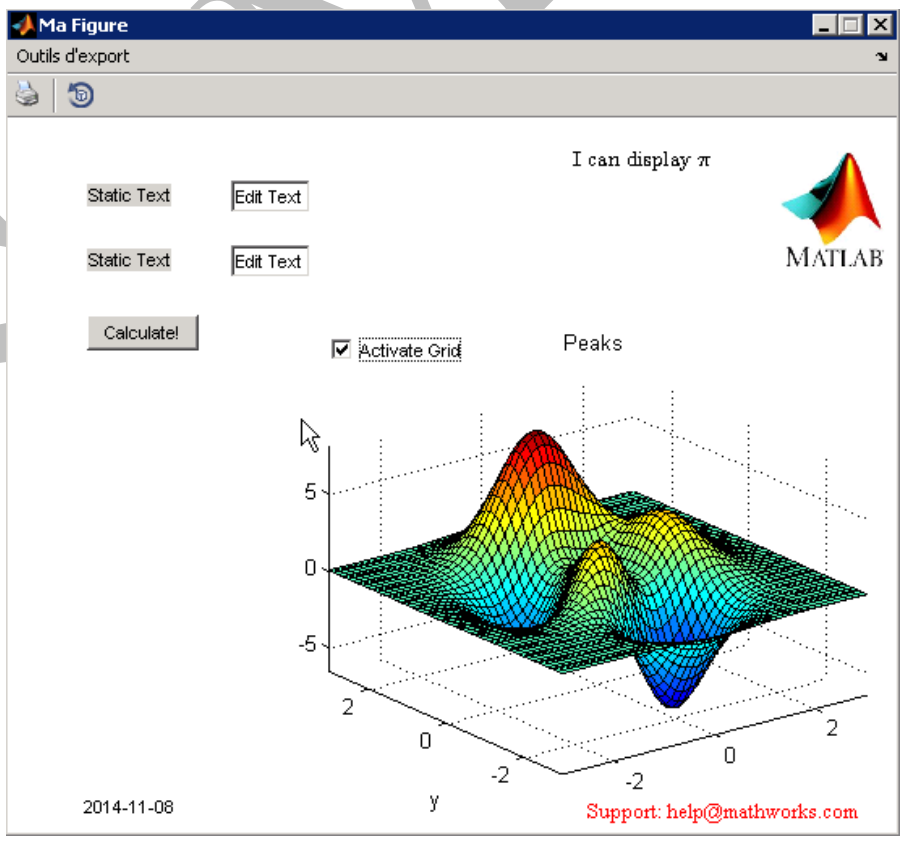
mon_GUI.m

```
156 -   set(handles.edit2,'String',sng_Value);
157
158
159     % --- Executes on button press in chk_Grid.
160     function chk_Grid_Callback(hObject, eventdata, handles)
161     % hObject    handle to chk_Grid (see GCBO)
162     % eventdata  reserved - to be defined in a future version of MATLAB
163     % handles    structure with handles and user data (see GUIDATA)
164
165     % Hint: get(hObject,'Value') returns toggle state of chk_Grid
166 -   chk_Grid_Status=get(hObject,'Value');
167 -   if chk_Grid_Status==1
168 -       axes(handles.pltChart);
169 -       grid on;
170 -   else
171 -       axes(handles.pltChart);
172 -       grid off;
173 -   end
```

And now if we execute the GUI we get:

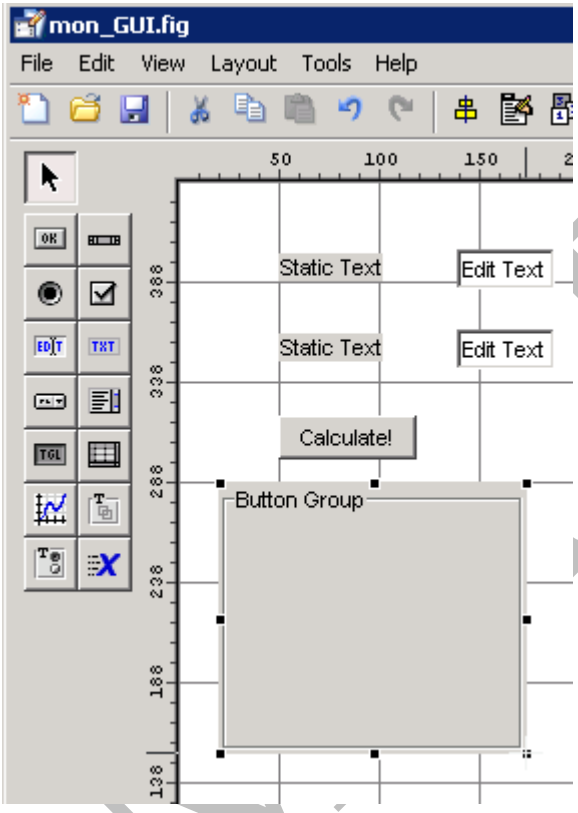


And with the checkbox activated:



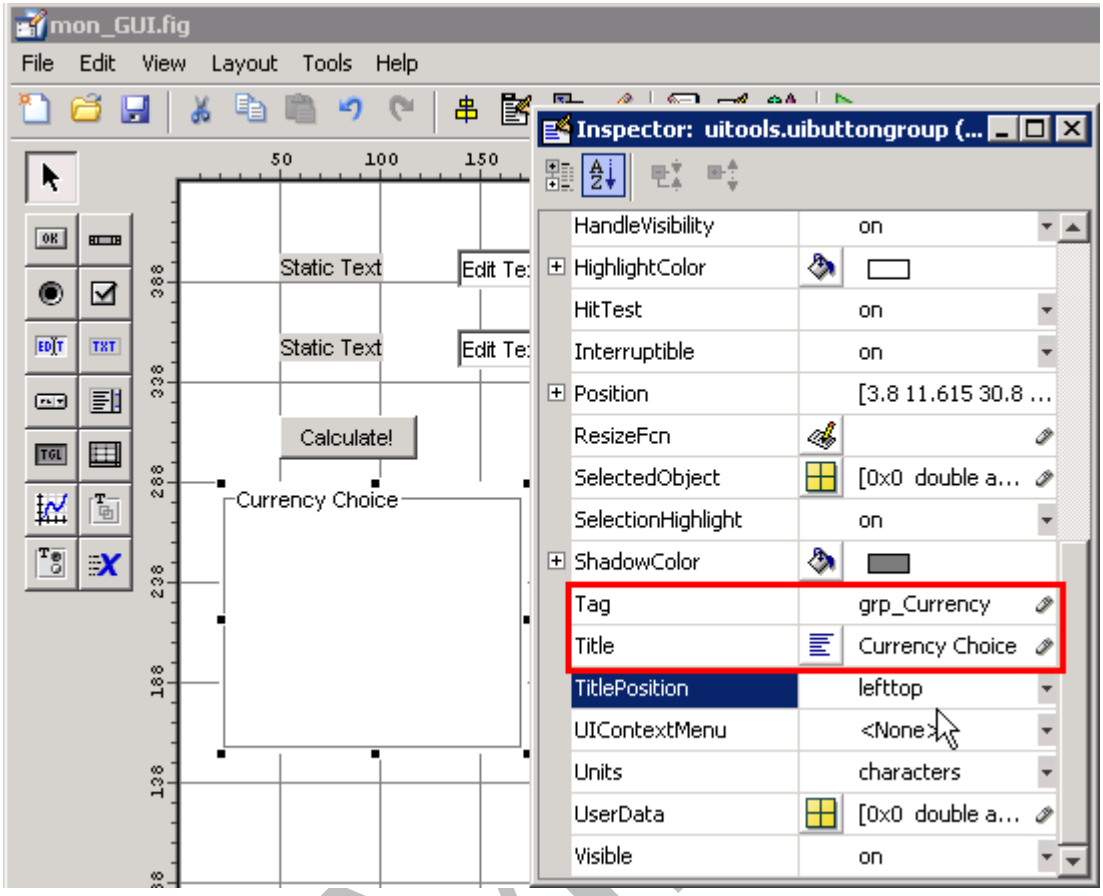
Manage radio buttons group

Another famous type or components are **Radio buttons**  . A good practice before inserting them is to add a **Buttons group**  before. So add this one first:

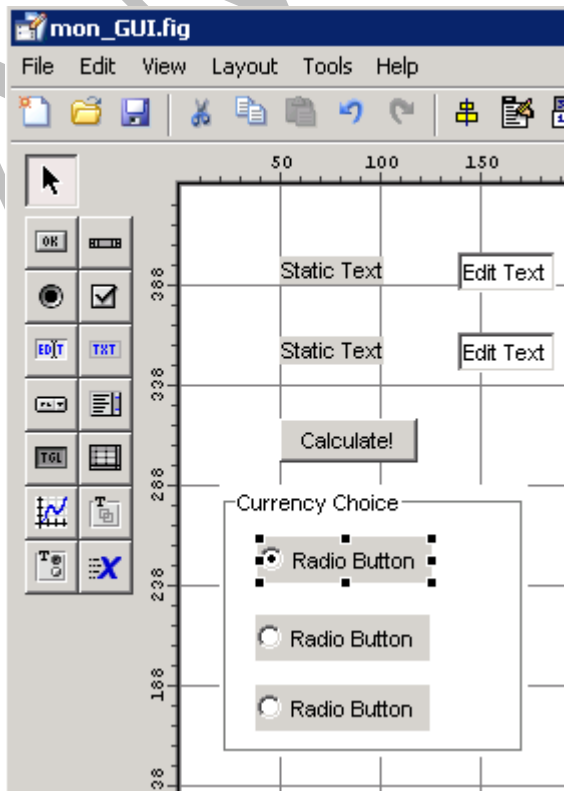


SAMPLE

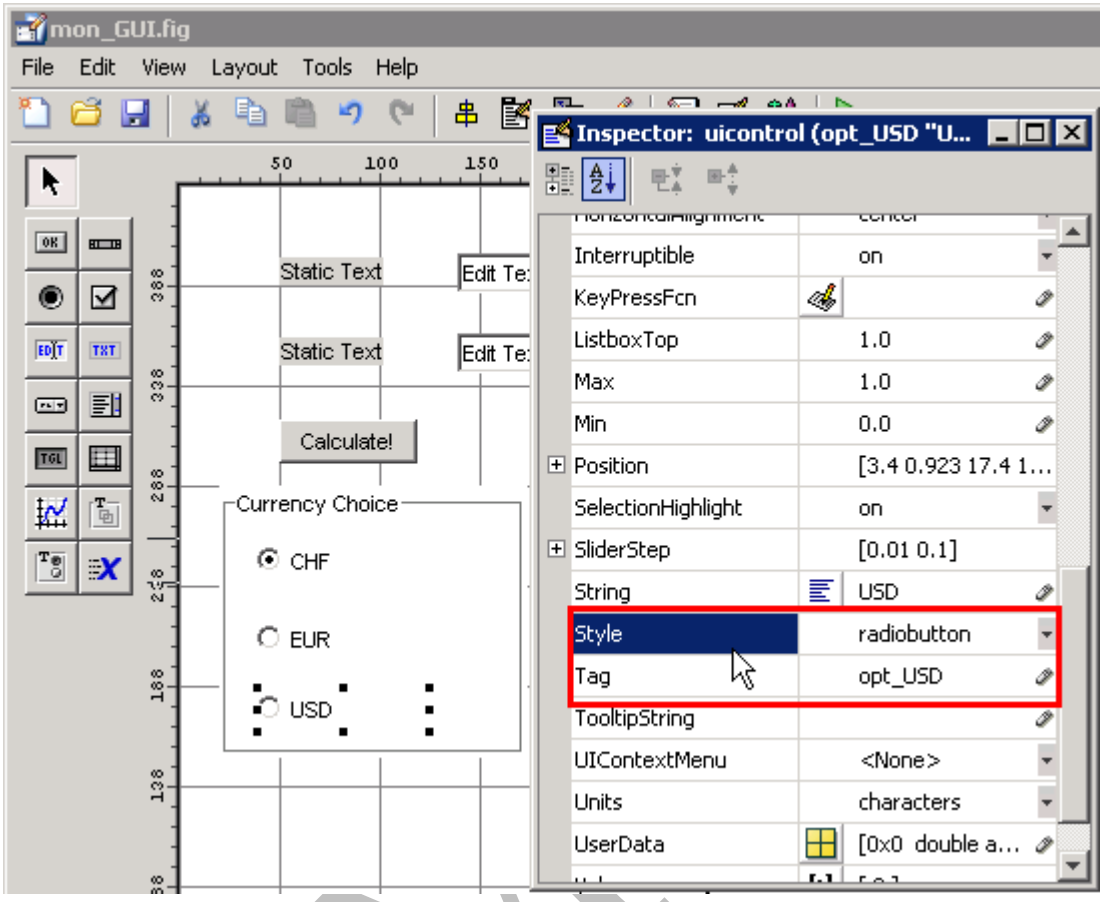
Change its **BackgroundColor**, **Title** and **Tag** properties:



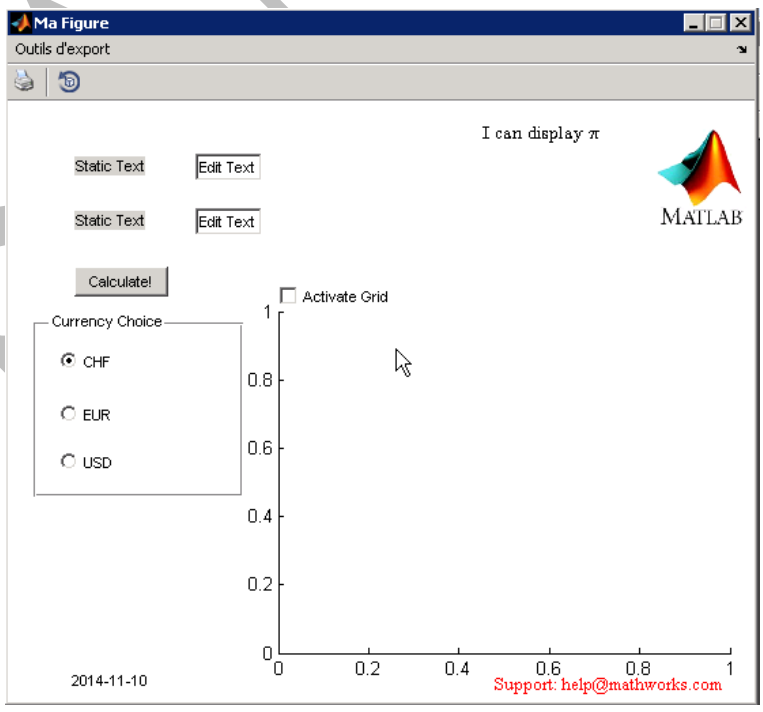
Once this is done add for example three radio buttons:



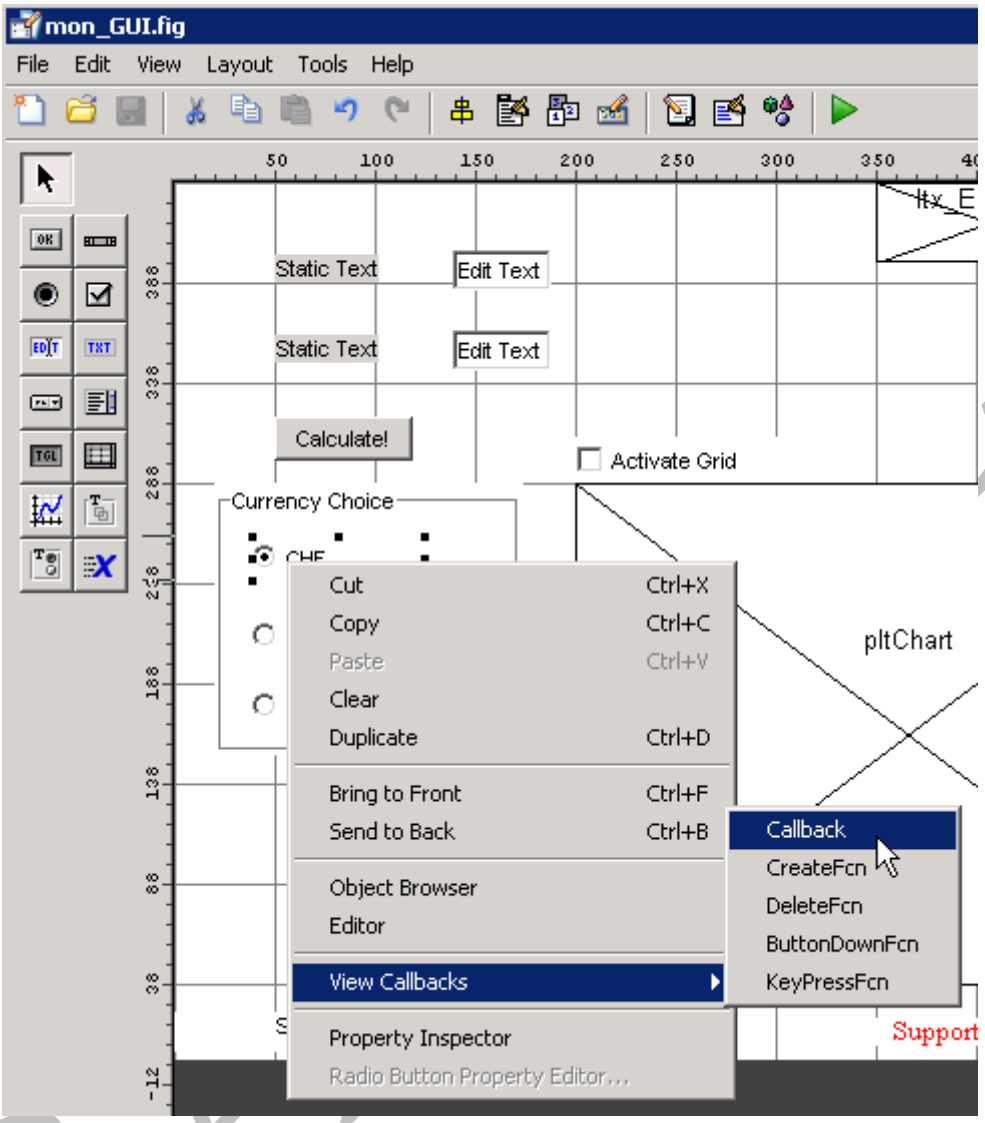
And also change their **BackgroundColor**, **Title** and **Tag** properties:



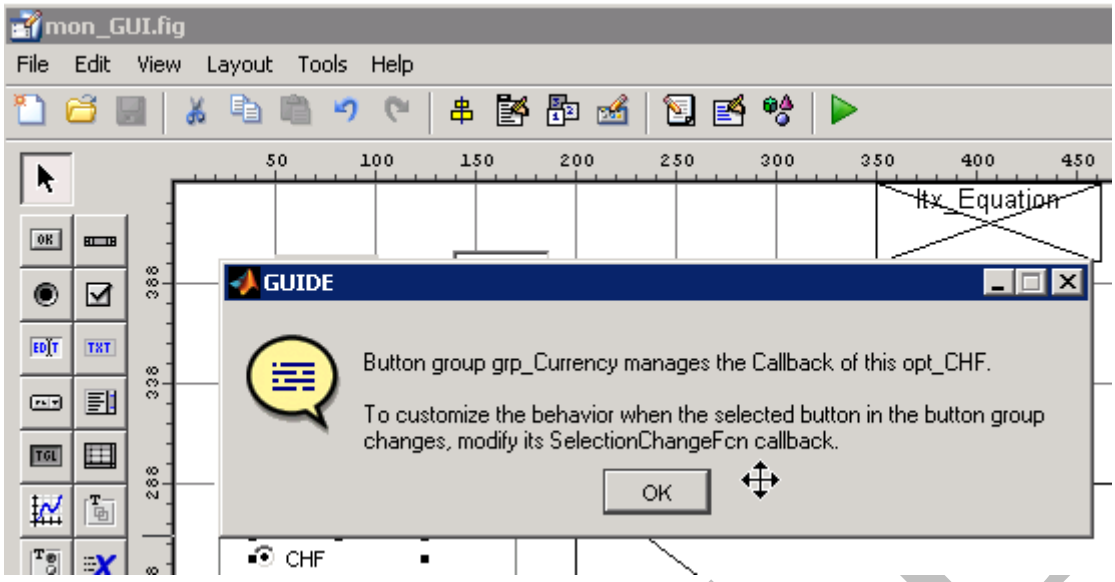
The first thing you can already see if you execute the GUI is that you can't activate all radio buttons at the same time because of the Button group area:



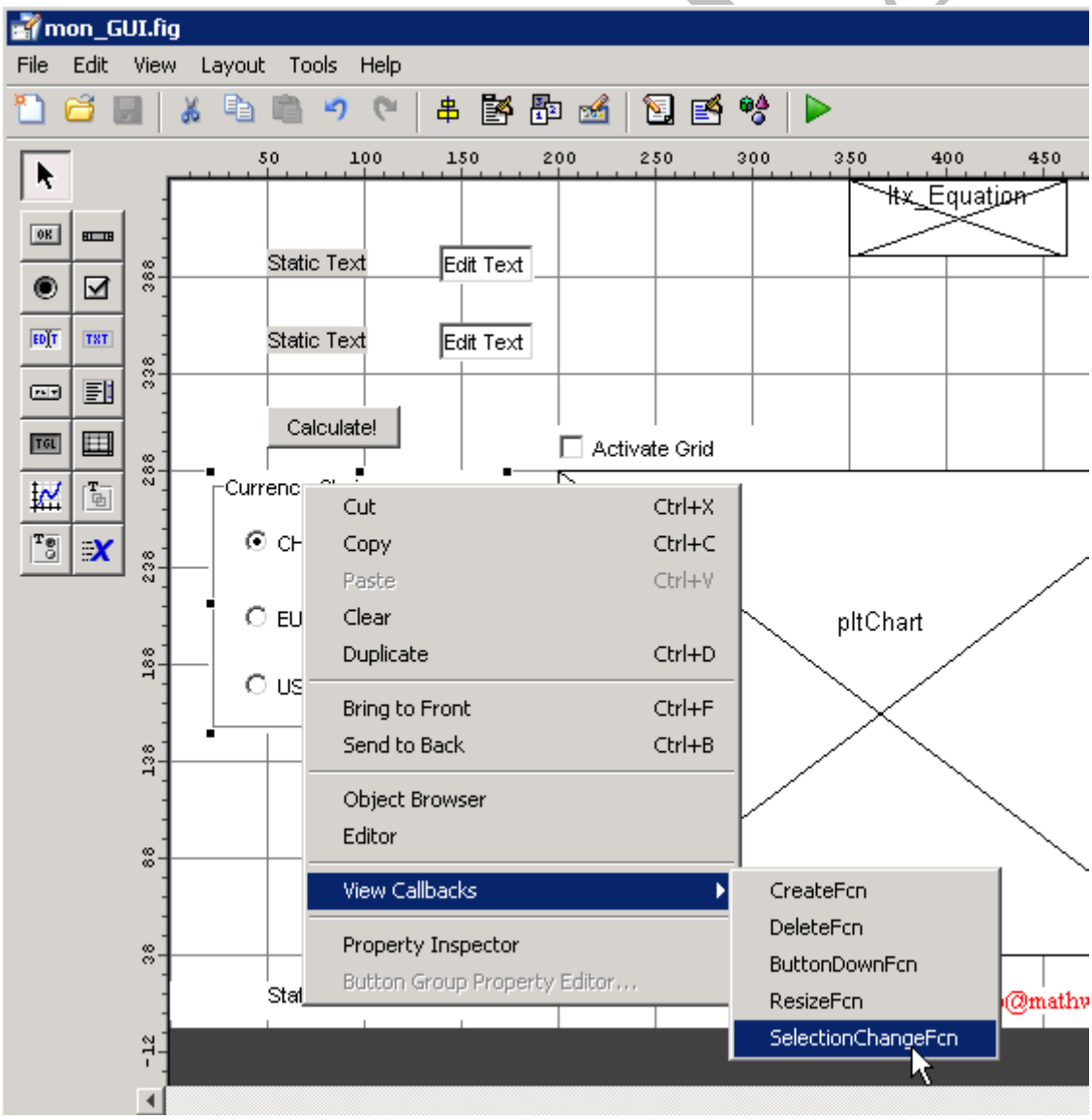
Now we will associate a Callback to the radiobuttons. Here we show only for the first one because the ideas are exactly the same for the others:



You will have:



You click on **OK** and we take as recommended the **SelectionChangeFcn** callback:

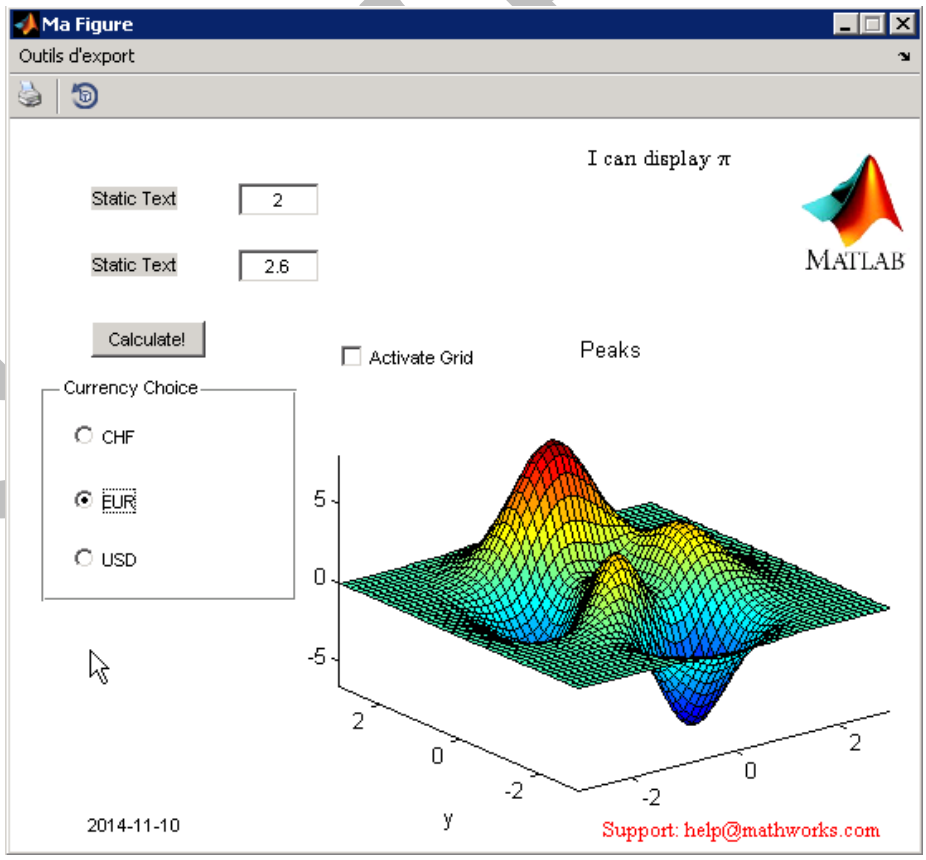


And we write:

mon_GUI.m

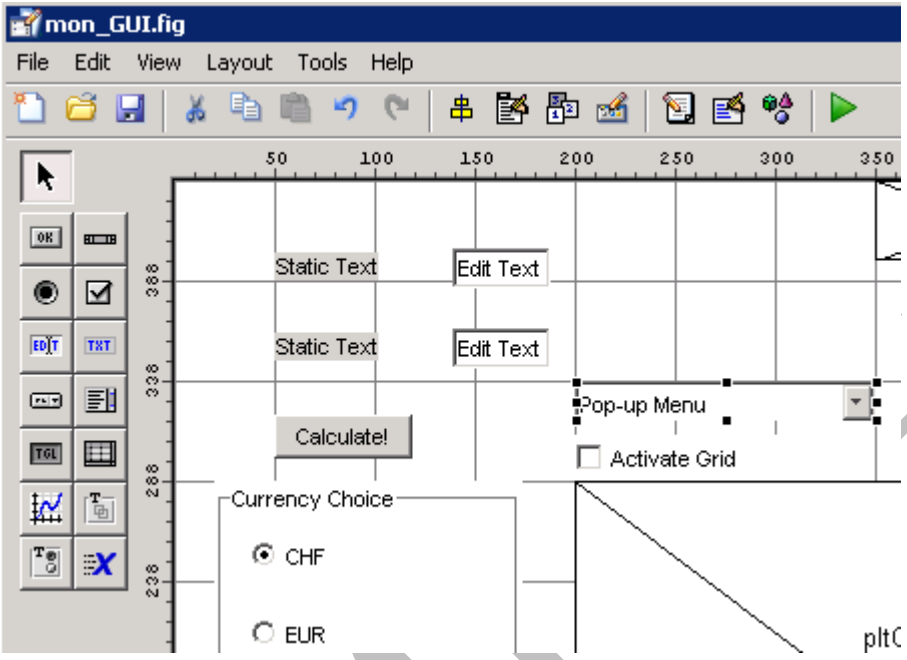
```
174
175
176     % --- Executes when selected object is changed in grp_Currency.
177     function grp_Currency_SelectionChangeFcn(hObject, eventdata, handles)
178     % hObject    handle to the selected object in grp_Currency
179     % eventdata  structure with the following fields (see UIBUTTONGROUP)
180     %   EventName: string 'SelectionChanged' (read only)
181     %   OldValue: handle of the previously selected object or empty if none was selected
182     %   NewValue: handle of the currently selected object
183     % handles    structure with handles and user data (see GUIDATA)
184 -   switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
185 -       case 'opt_CHF'
186 -           sng_Value=str2double(get(handles.edit1,'String'))*1.2;
187 -       case 'opt_EUR'
188 -           sng_Value=str2double(get(handles.edit1,'String'))*1.3;
189 -       case 'opt_USD'
190 -           sng_Value=str2double(get(handles.edit1,'String'))*1;
191 -   end
192 -   set(handles.edit2,'String',sng_Value);
193
```

and so we get the expected result:

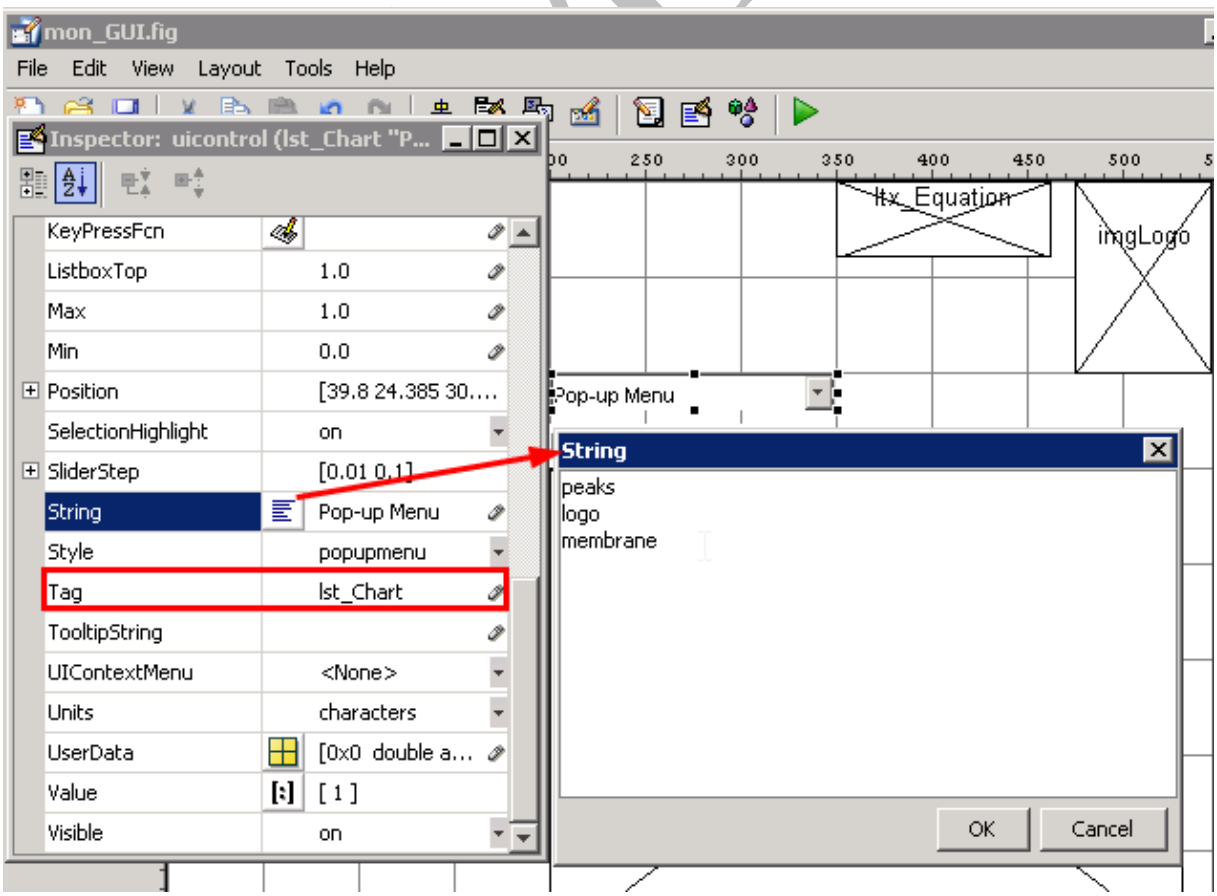


Manage pop-up menu and get selected value

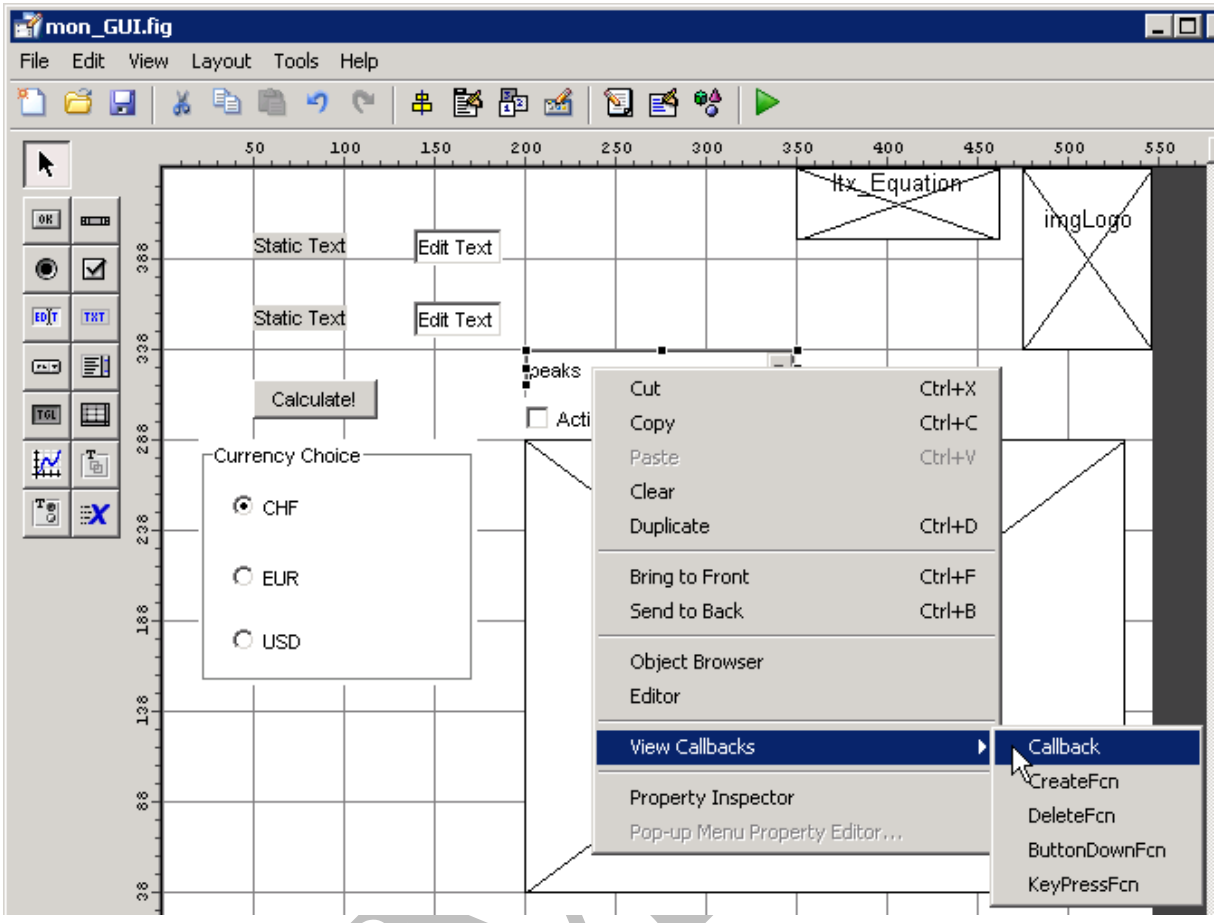
We want to choose in a static combo-box a selection of charts. For this purpose add pop-up menu  control in the GUI:



Open its properties and change the **String** and **Tag** properties:



Once done, open its **Callback**:

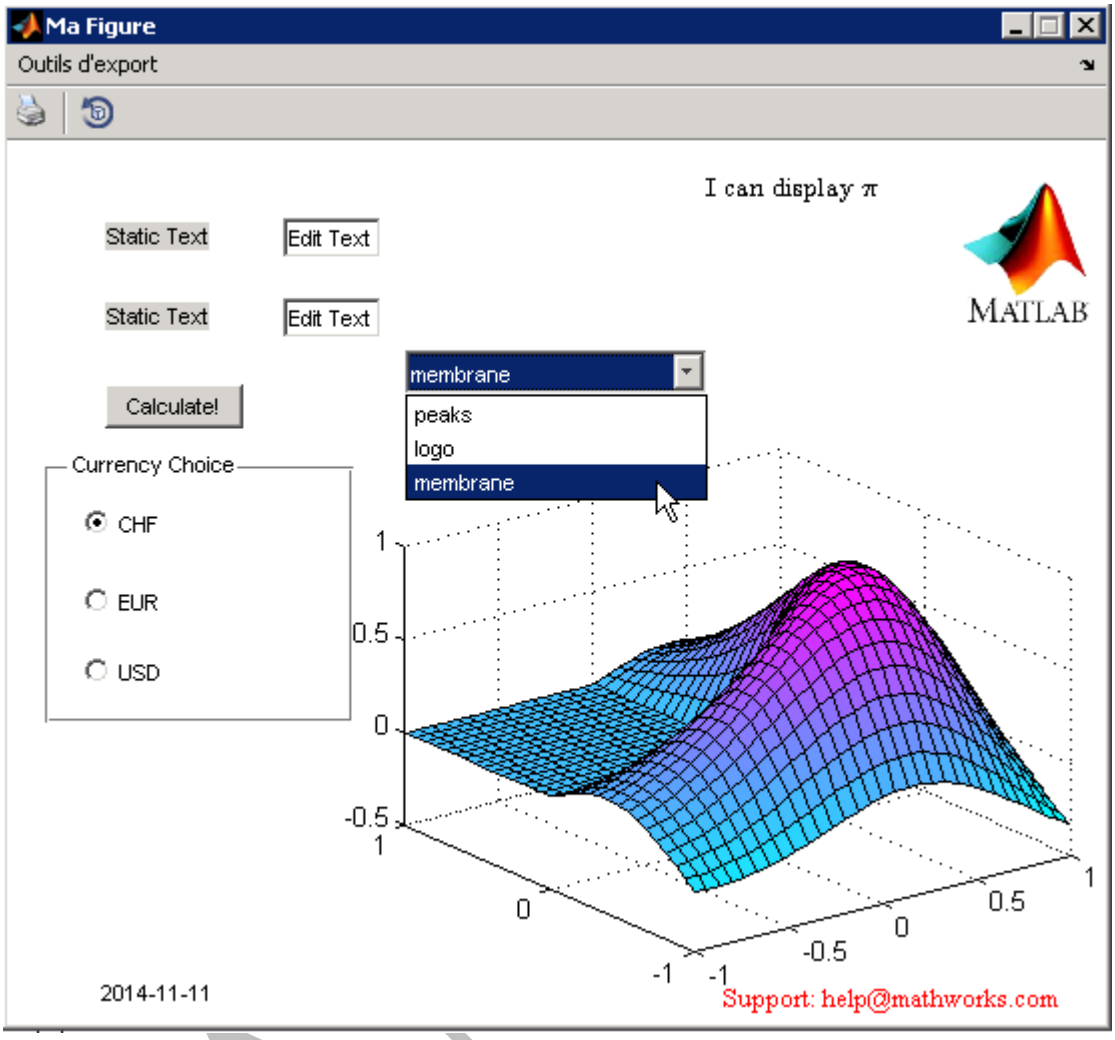


and add the following code:

mon_GUI.m

```
194
195     % --- Executes on selection change in lst_Chart.
196     function lst_Chart_Callback(hObject, eventdata, handles)
197     % hObject    handle to lst_Chart (see GCBO)
198     % eventdata  reserved - to be defined in a future version of MATLAB
199     % handles    structure with handles and user data (see GUIDATA)
200
201     % Hints: contents = cellstr(get(hObject,'String')) returns lst_Chart contents as cell array
202     %        contents{get(hObject,'Value')} returns selected item from lst_Chart
203 -   val=get(hObject,'Value');
204 -   str=get(hObject,'String');
205 -   switch str{val}
206 -       case 'peaks'
207 -           axes(handles.pltChart);
208 -           peaks;
209 -       case 'membrane'
210 -           axes(handles.pltChart);
211 -           membrane;
212 -       case 'logo'
213 -           axes(handles.pltChart);
214 -           logo;
215 -   end
```

Now if as always we save the changes and run the GUI we get:



And if you want to fill dynamically the pop-up menu, here is what you need to use for example when the GUI loads:

mon_GUI.m

```
47    % --- Executes just before monGUI is made visible.
48    function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.
50
51    % Choose default command line output for monGUI
52    handles.output = hObject;
53
54    % Update handles structure
55    guidata(hObject, handles);
56
57    set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59    axes(handles.ltx_Equation);
60    axis off;
61    text(0,0.5,'I can display $\pi$','interpreter','latex',...
62    'horiz','left','vert','middle')
63
64    axes(handles.imgLogo);
65    imshow('matlab.jpg');
66    axes(handles.pltChart);
67    peaks;
68    grid off;
69    string_list{1}='peaks';
70    string_list{2}='logo';
71    string_list{3}='membrane';
72    set(handles.lst_Chart,'String',string_list);
```

and for sure you can automate this with a loop! Or if you prefer:

SAMPLE

mon_GUI.m*

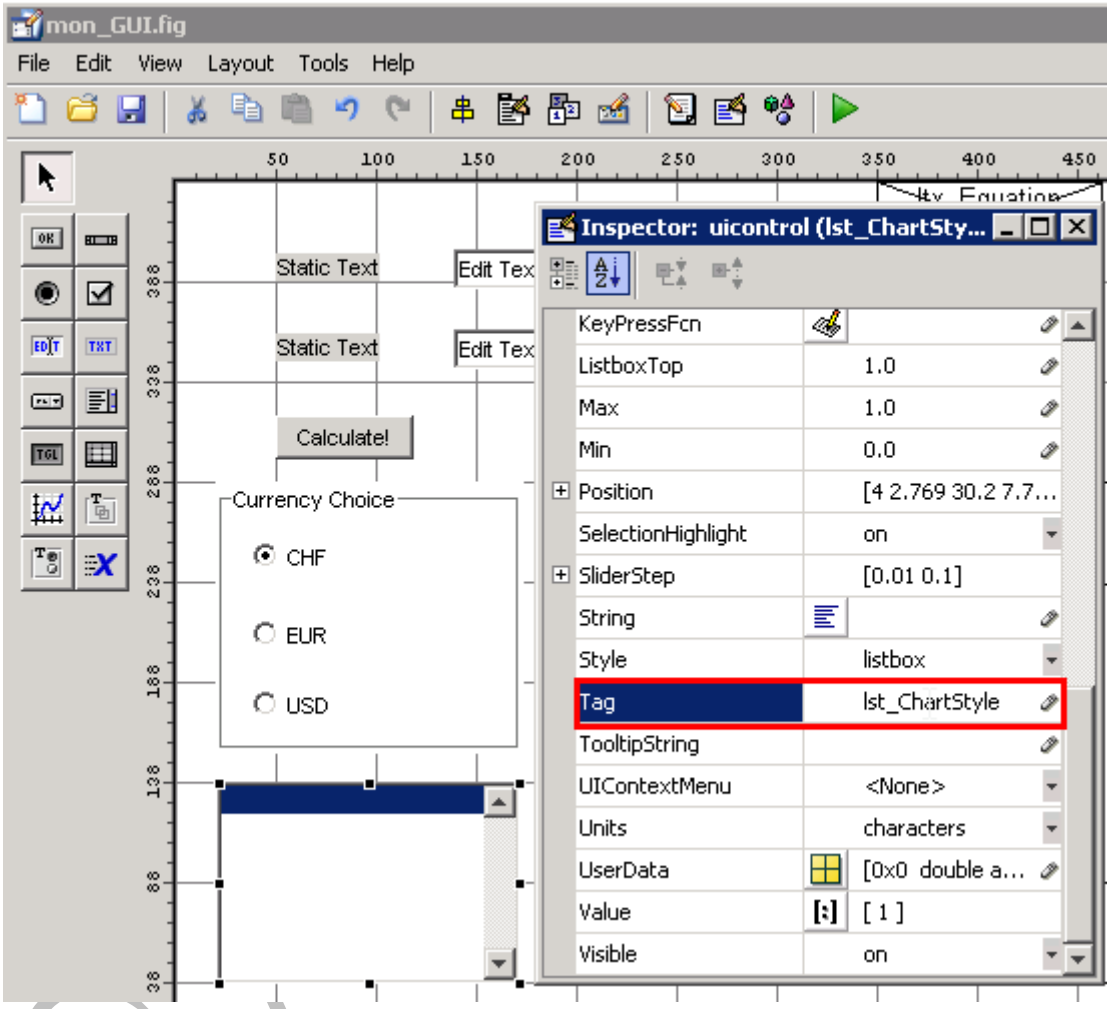
```
47      % --- Executes just before monGUI is made visible.
48      function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49      % This function has no output args, see OutputFcn.
50
51      % Choose default command line output for monGUI
52      handles.output = hObject;
53
54      % Update handles structure
55      guidata(hObject, handles);
56
57      set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59      axes(handles.ltx_Equation);
60      axis off;
61      text(0,0.5,'I can display $\pi$','interpreter','latex',...
62      'horiz','left','vert','middle')
63
64      axes(handles.imgLogo);
65      imshow('matlab.jpg');
66      axes(handles.pltChart);
67      peaks;
68      grid off;
69      set(handles.lst_Chart,'String',{'peaks','logo','membrane'});
```

So when we run the GUI we also get the same thing as before!

SAMPLE

Manage multiple selection List box

For the multiple selection List box  we will directly make a dynamic one (we already saw before how to manage static stuff). First create such a component following exactly the same steps as we did for the pop-menu (simple list box):



Once this is done we fill it dynamically on the GUI open event callback:

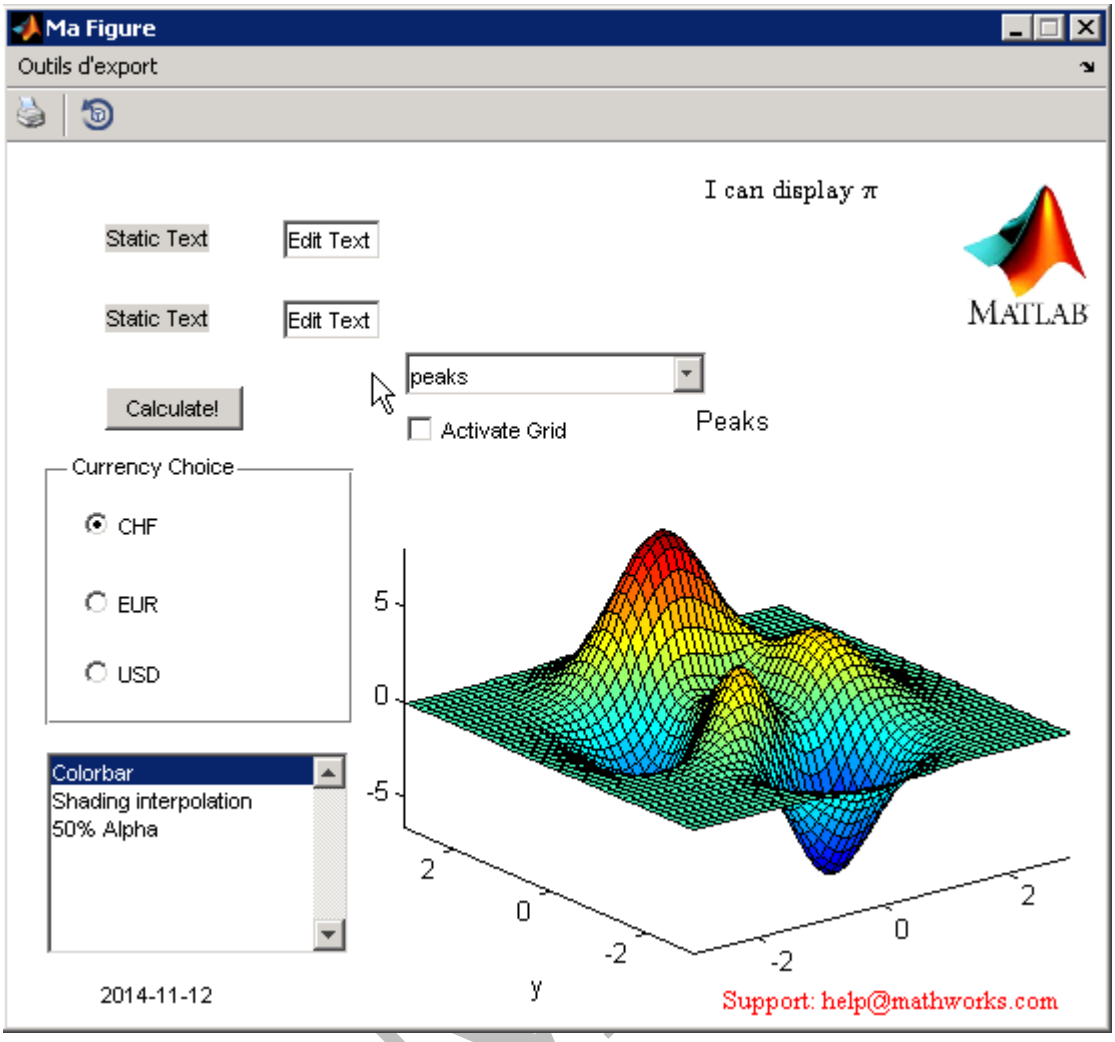
EDITOR PUBLISH VIEW

mon_GUI.m

```
47    % --- Executes just before monGUI is made visible.
48    function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.
50
51    % Choose default command line output for monGUI
52 -  handles.output = hObject;
53
54    % Update handles structure
55 -  guidata(hObject, handles);
56
57 -  set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59 -  axes(handles.ltx_Equation);
60 -  axis off;
61 -  text(0,0.5,'I can display $\pi$','interpreter','latex',...
62    'horiz','left','vert','middle')
63
64 -  axes(handles.imgLogo);
65 -  imshow('matlab.jpg');
66 -  axes(handles.pltChart);
67 -  peaks;
68 -  grid off;
69 -  set(handles.lst_Chart,'String',{'peaks','logo','membrane'});
70 -  set(handles.lst_ChartStyle,'String',{'Colorbar','Shading interpolation','50% Alpha'});
```

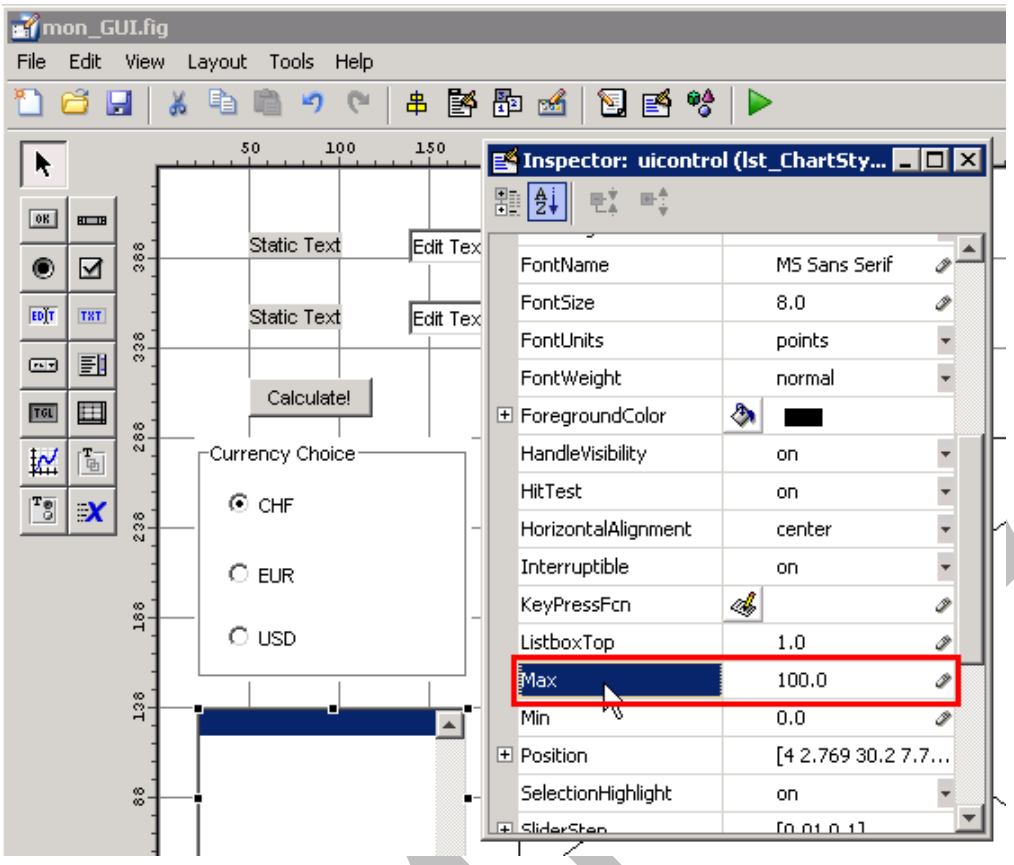
If we already run the GUI we get:

SAMPLE

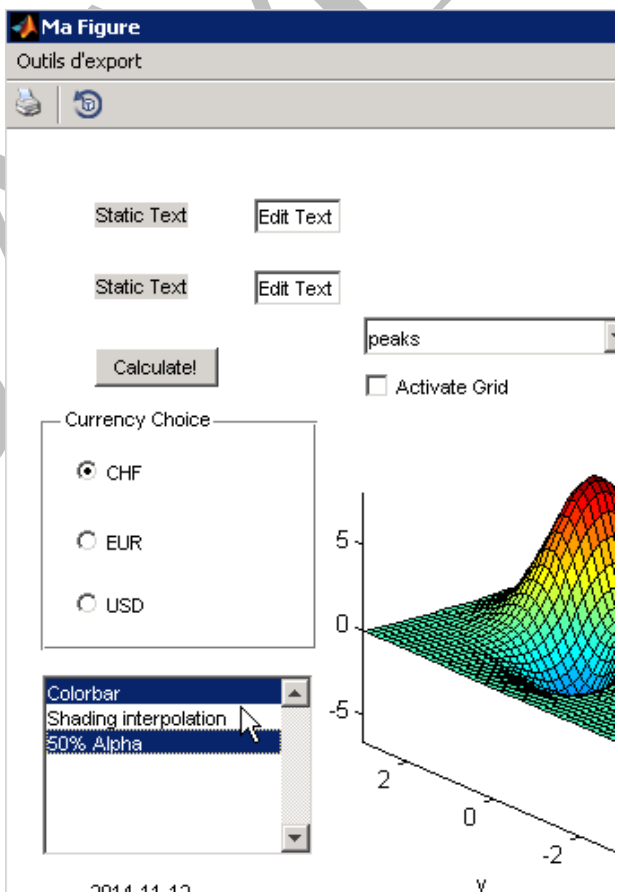


To make multiple selections possible you have to change the **Min** and **Max** properties. The difference corresponds to the number of possible selections. For example, even if we have only three choices we can put:

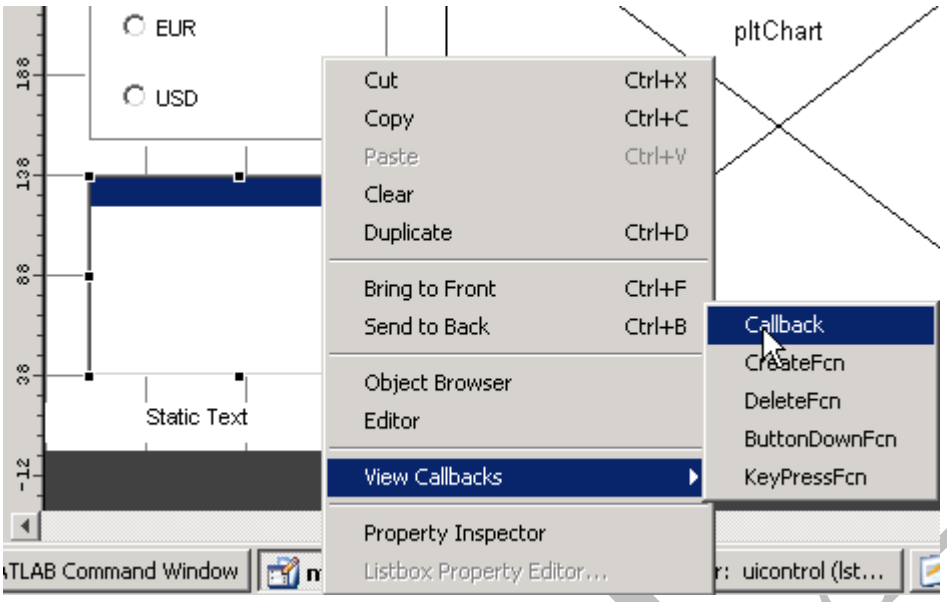
SAMPLE


mon_GUI.fig
File Edit View Layout Tools Help
Inspector: uicontrol (lst_ChartSty...
Static Text
Edit Tex
Static Text
Edit Tex
Calculate!
Currency Choice
CHF
EUR
USD
FontName MS Sans Serif
FontSize 8.0
FontUnits points
FontWeight normal
ForegroundColor
HandleVisibility on
HitTest on
HorizontalAlignment center
Interruptible on
KeyPressFcn
ListboxTop 1.0
Max 100.0
Min 0.0
Position [4 2.769 30.2 7.7...
SelectionHighlight on

So:


Ma Figure
Outils d'export
Static Text
Edit Text
Static Text
Edit Text
peaks
Calculate!
Activate Grid
Currency Choice
CHF
EUR
USD
Colorbar
Shading interpolation
50% Alpha
5
0
-5
2
0
-2

Seems to work well! We continue now to define a callback on this list box:

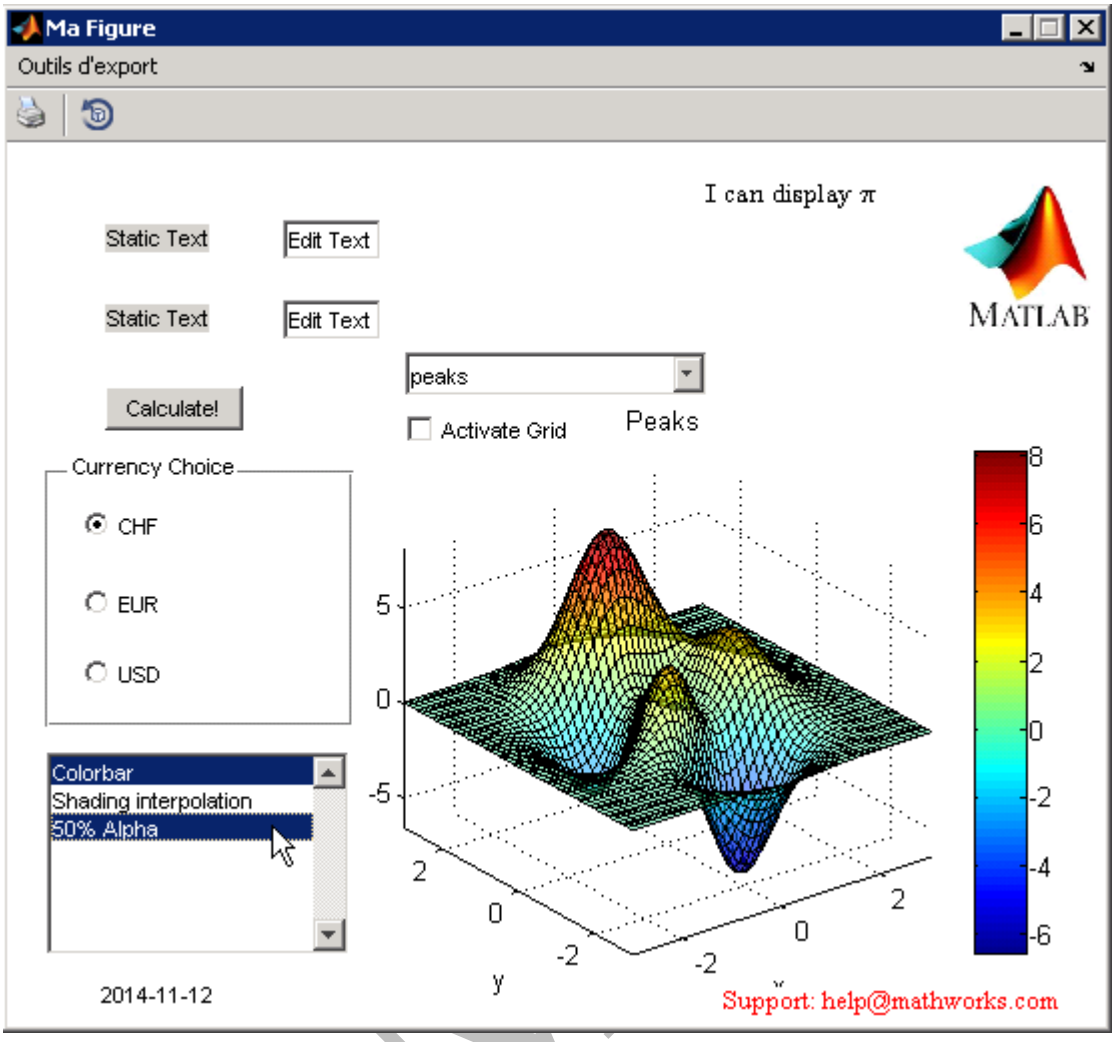


And now we write the callback for the List Box:

mon_GUI.m

```
227   % --- Executes on selection change in lst_ChartStyle.
228   function lst_ChartStyle_Callback(hObject, eventdata, handles)
229   % hObject    handle to lst_ChartStyle (see GCBO)
230   % eventdata  reserved - to be defined in a future version of MATLAB
231   % handles    structure with handles and user data (see GUIDATA)
232
233   % Hints: contents = cellstr(get(hObject,'String')) returns lst_ChartStyle contents as cell array
234   %        contents{get(hObject,'Value')} returns selected item from lst_ChartStyle
235 - str = get(hObject,{'string','value'});
236 - str = str{1}(str{2});
237 - peaks;
238 - if find(ismember(str,'Colorbar'))
239 -     axes(handles.pltChart);
240 -     colorbar;
241 - end
242 - if find(ismember(str,'Shading interpolation'))
243 -     axes(handles.pltChart);
244 -     shading interp;
245 - end
246 - if find(ismember(str,'50% Alpha'))
247 -     axes(handles.pltChart);
248 -     alpha(0.5);
249 - end
```

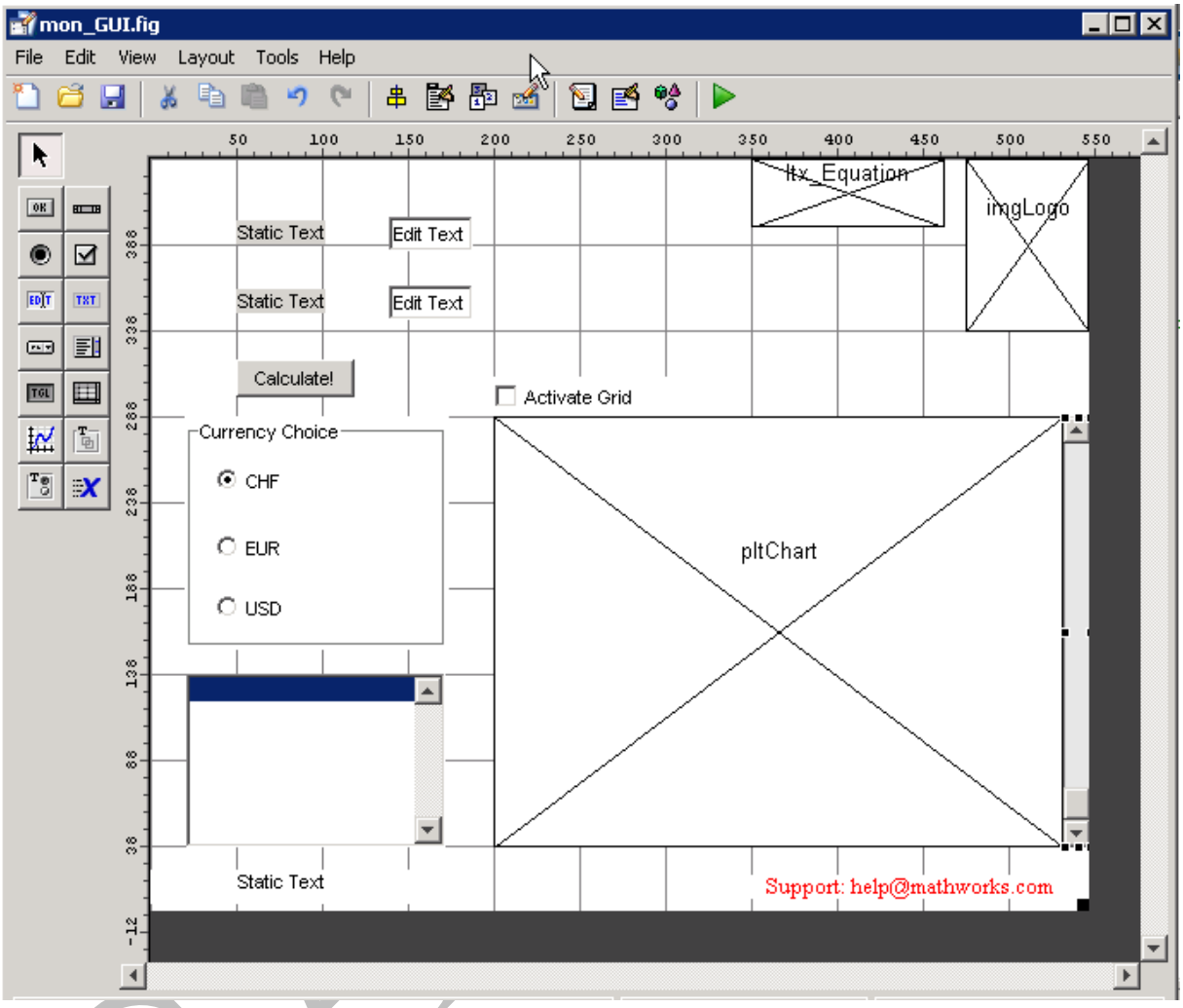
This works well as we can see it:


Ma Figure
Outils d'export
I can display π
Static Text
Edit Text
Static Text
Edit Text
Calculate!
peaks
Activate Grid
Peaks
Currency Choice
CHF
EUR
USD
Colorbar
Shading interpolation
50% Alpha
MATLAB
2014-11-12
Support: help@mathworks.com

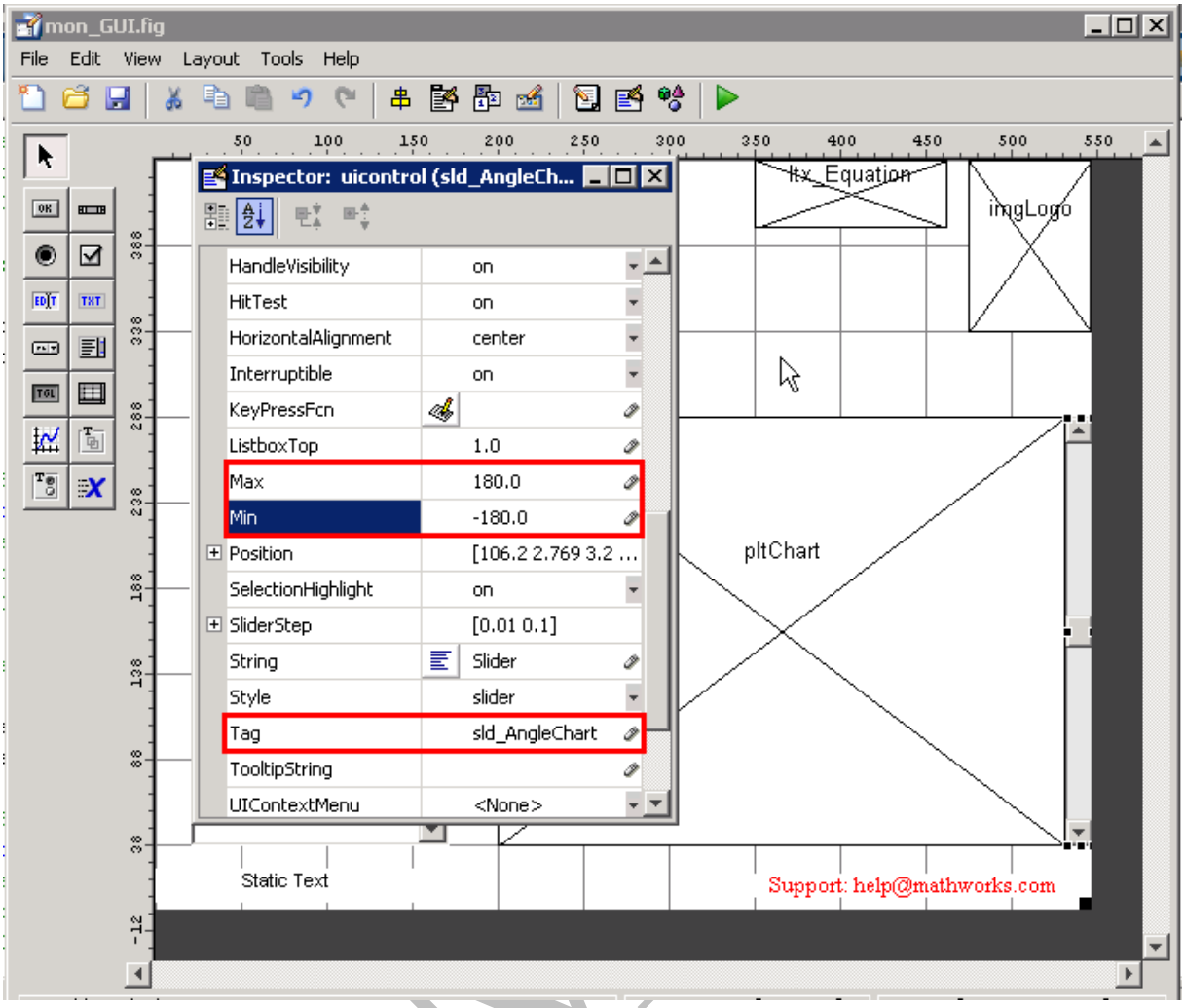
SAM

Manage sliders

Sliders  are very useful for all sort of simulations or chart manipulation. Let us have a look of how to use them. Add one right from the chart as following:



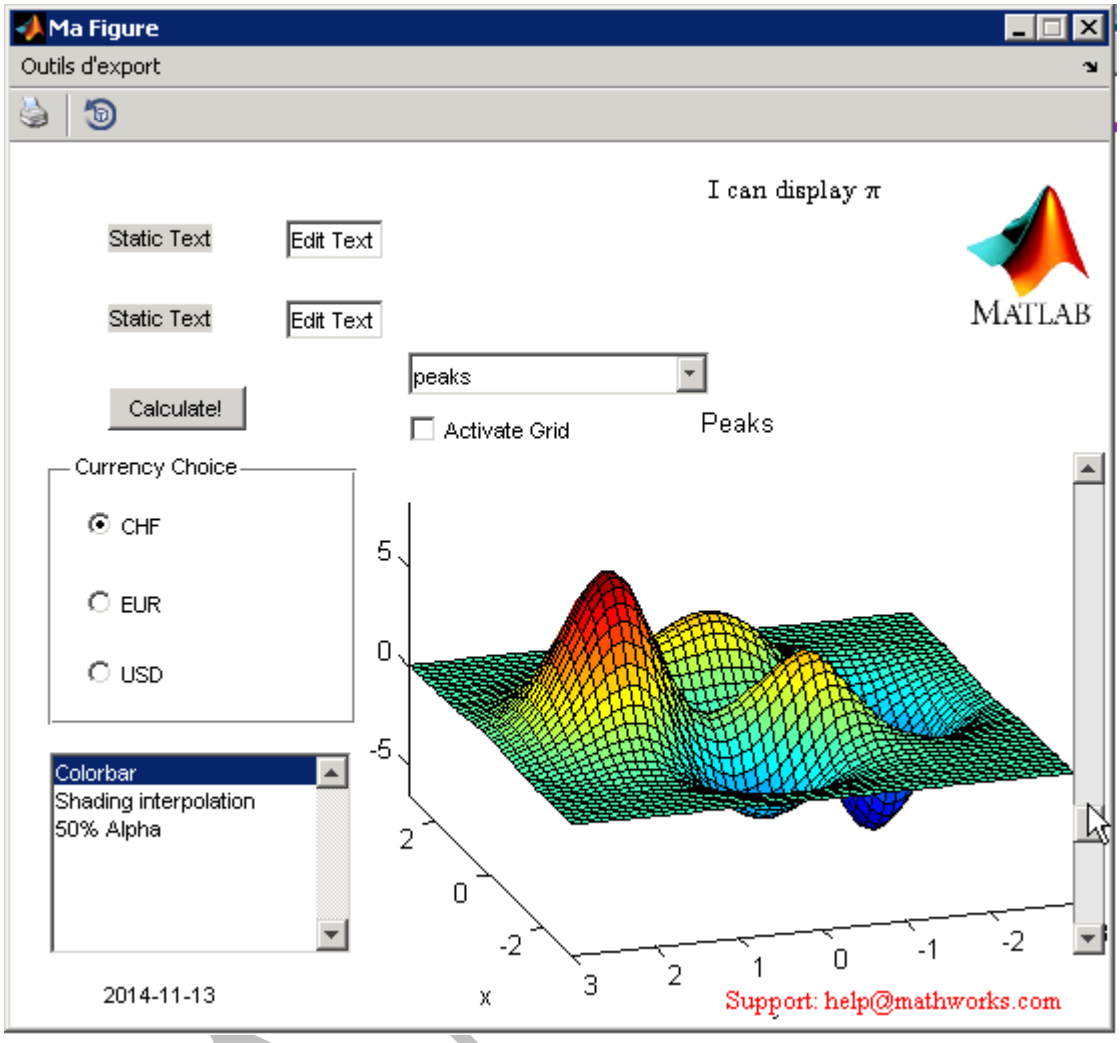
Open the property inspector and change the **Tag**, **Max** and **Min** properties:



Now we open the callback of the slider to write:

```
mon_GUI.m
264
265     % --- Executes on slider movement.
266     function sld_AngleChart_Callback(hObject, eventdata, handles)
267     % hObject    handle to sld_AngleChart (see GCBO)
268     % eventdata  reserved - to be defined in a future version of MATLAB
269     % handles    structure with handles and user data (see GUIDATA)
270
271     % Hints: get(hObject,'Value') returns position of slider
272     %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
273 -   axes(handles.pltChart);
274 -   view(get(hObject,'Value'), 30);
275
```

And if we run the GUI you can check that the slider works well:

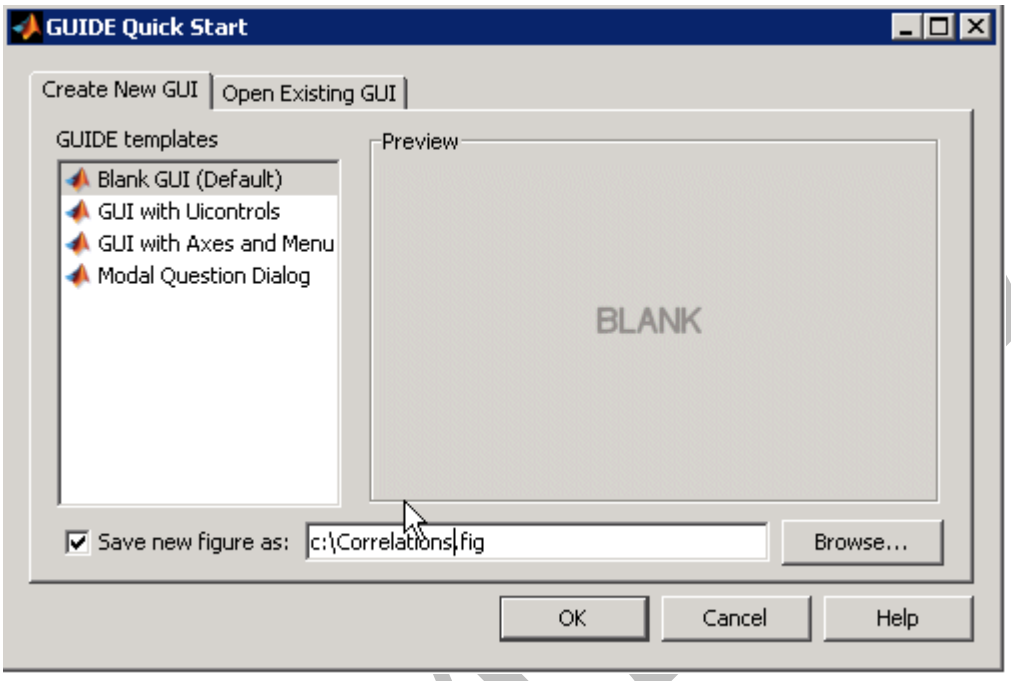


SAMPLE

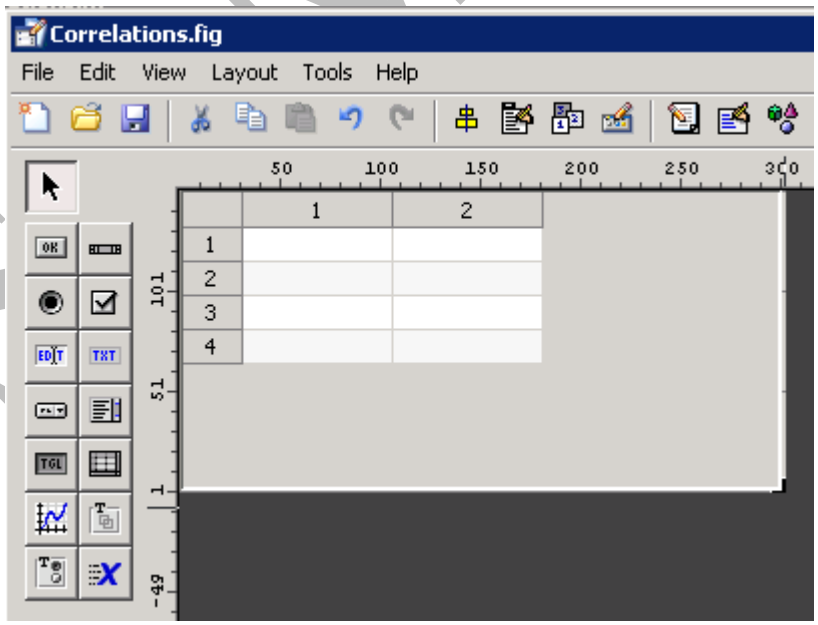
Manage tables

We can do a lot of things with MATLAB GUI tables: tables of values, tables of values with check boxes, tables of listboxes, etc. Here we will focus only on the basics.

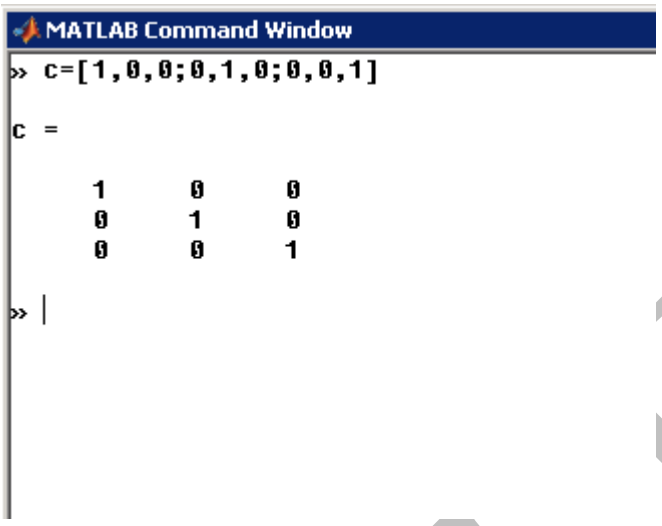
First let us create a Blank GUI:



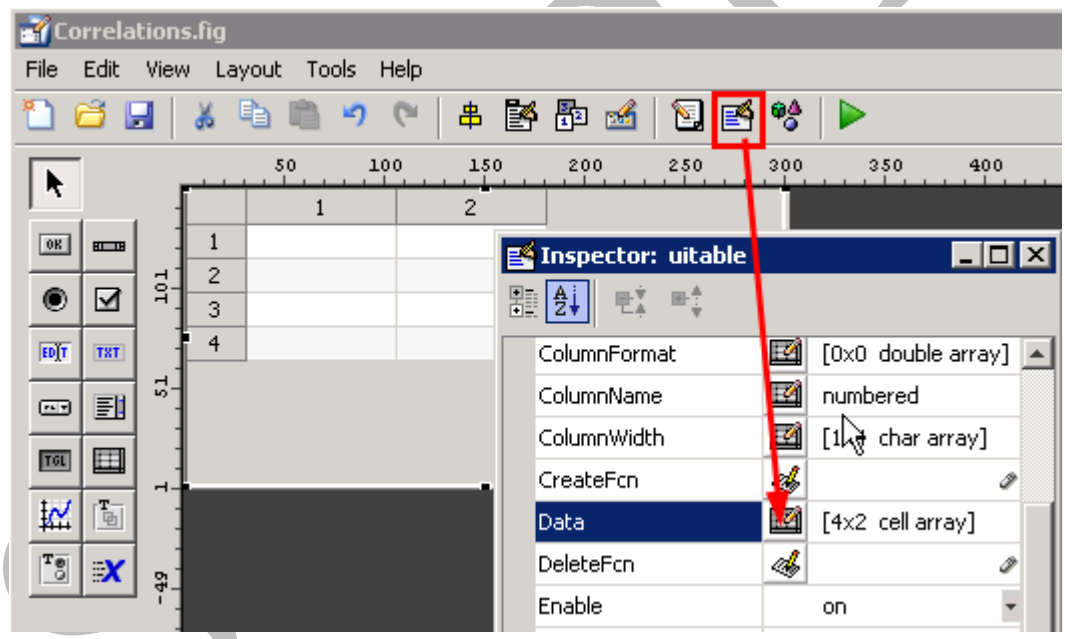
Afterwards in the toolbox pane we use the following button  to draw a table. And we get:



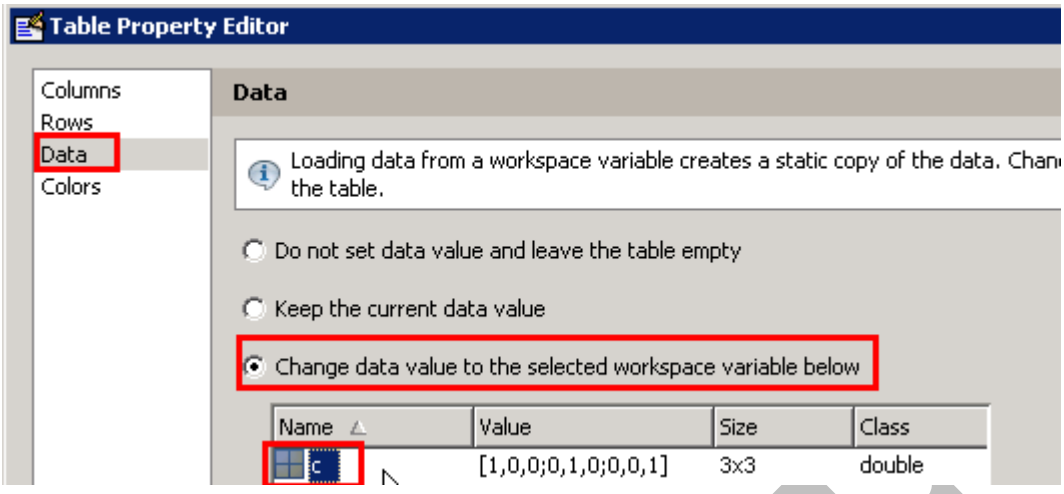
Now in the Command Windows, let us consider that we have the following variable in the memory:



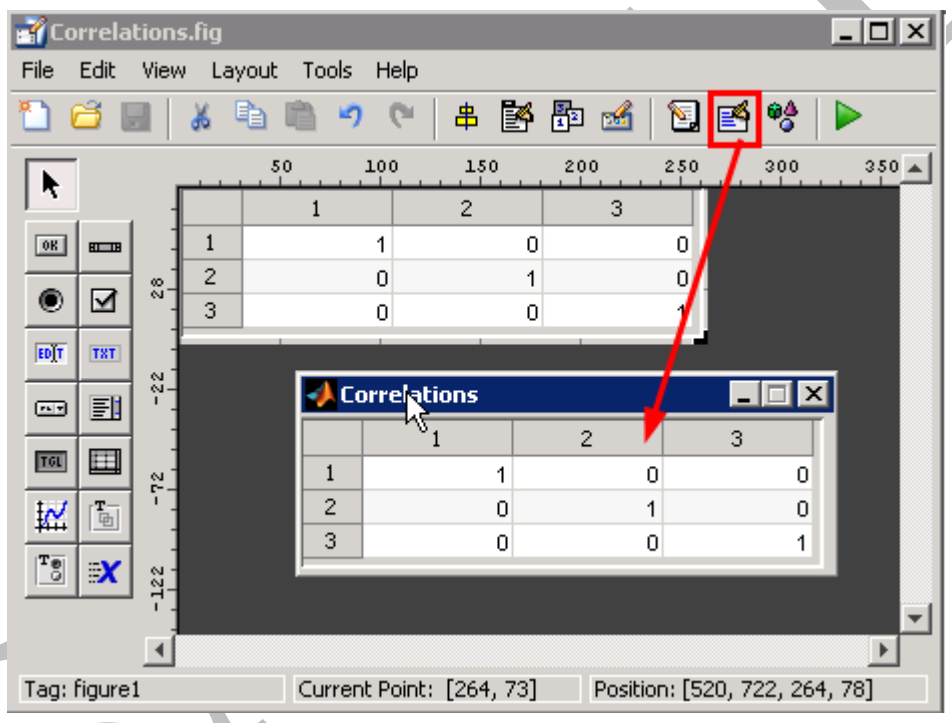
To connect our table to this variable we click on the following buttons:



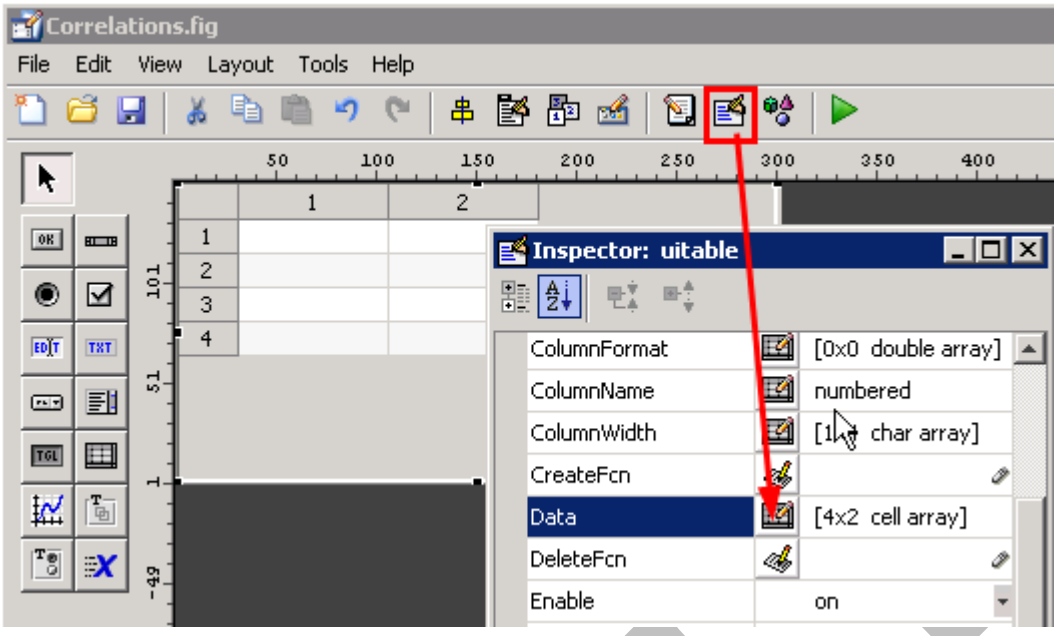
A wizard the appears:



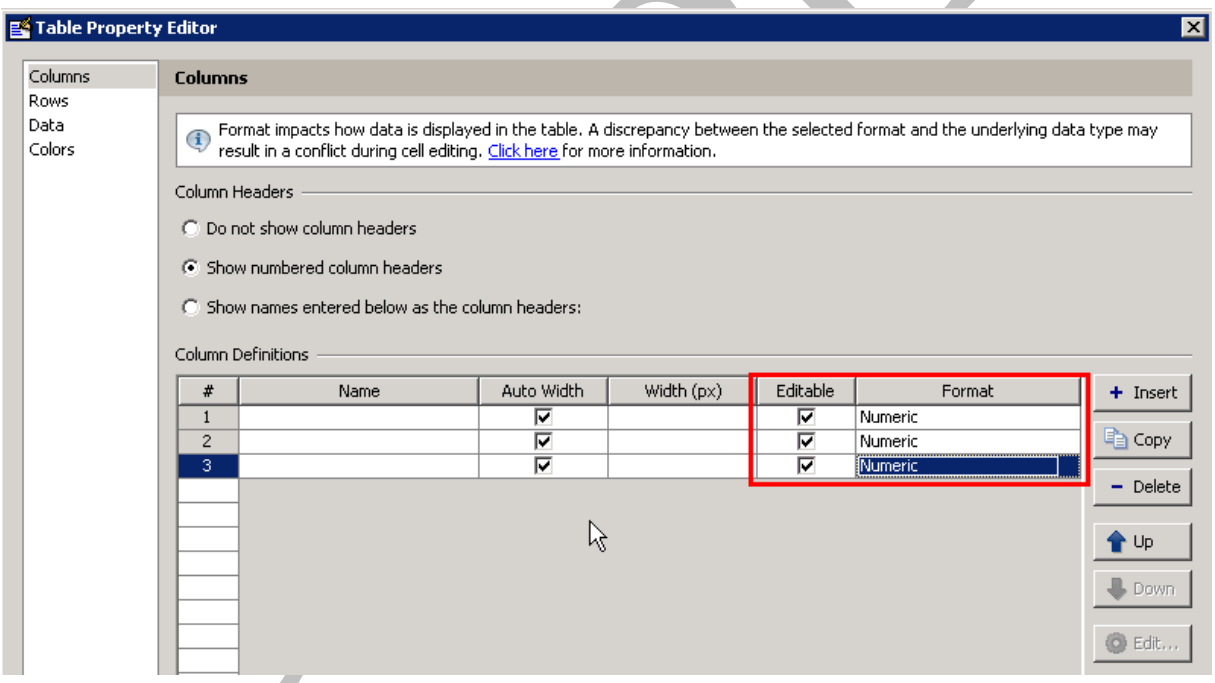
after validation, we get:



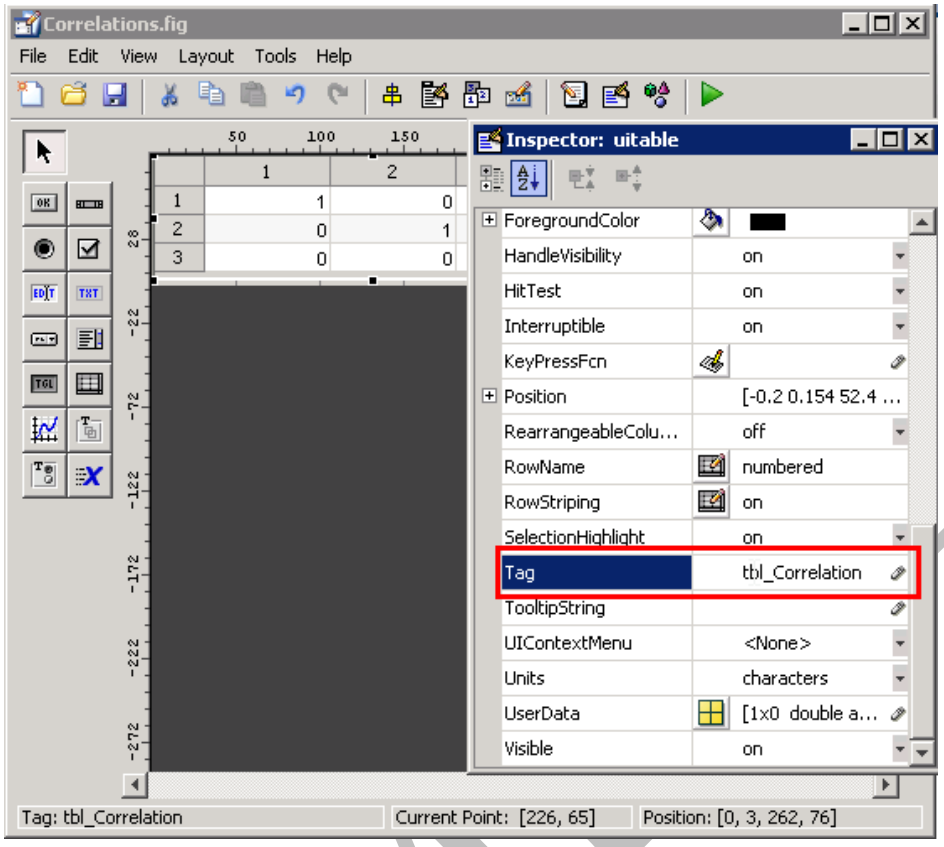
Ok the result is nice but… we want to make the cells editable!!! For this purpose we go back in the Wizard:



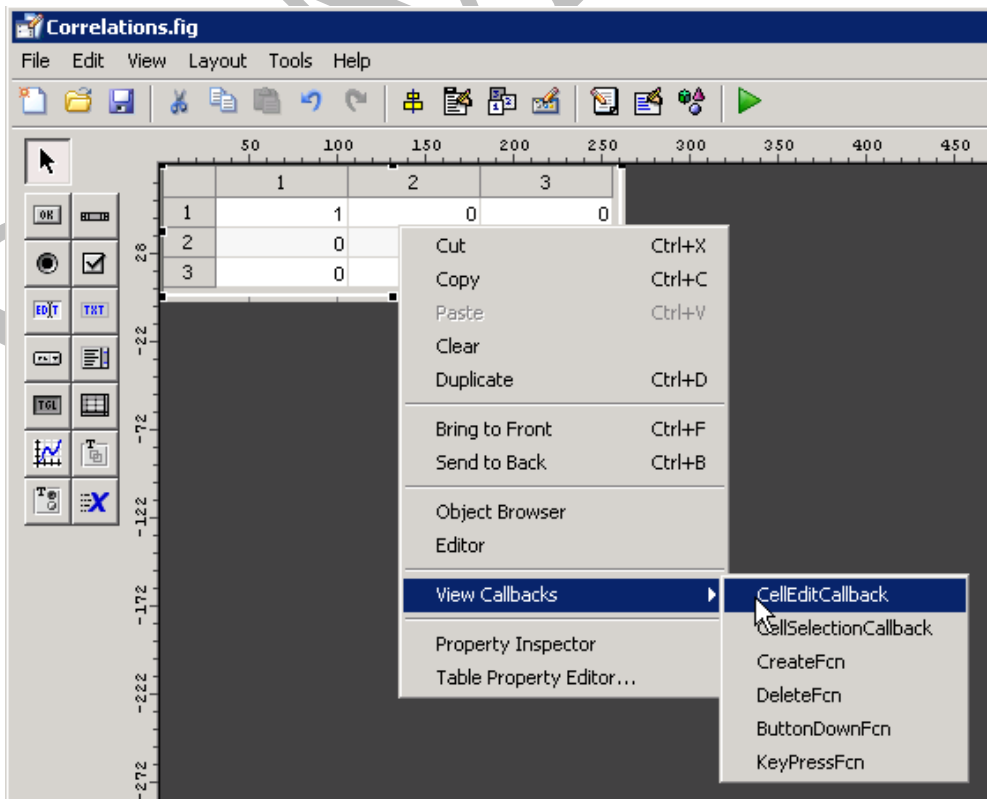
to tick the corresponding *Editable* column:



and in the Inspector we give an **Tag** name to our table:



After we need to edit the Callback of the table:

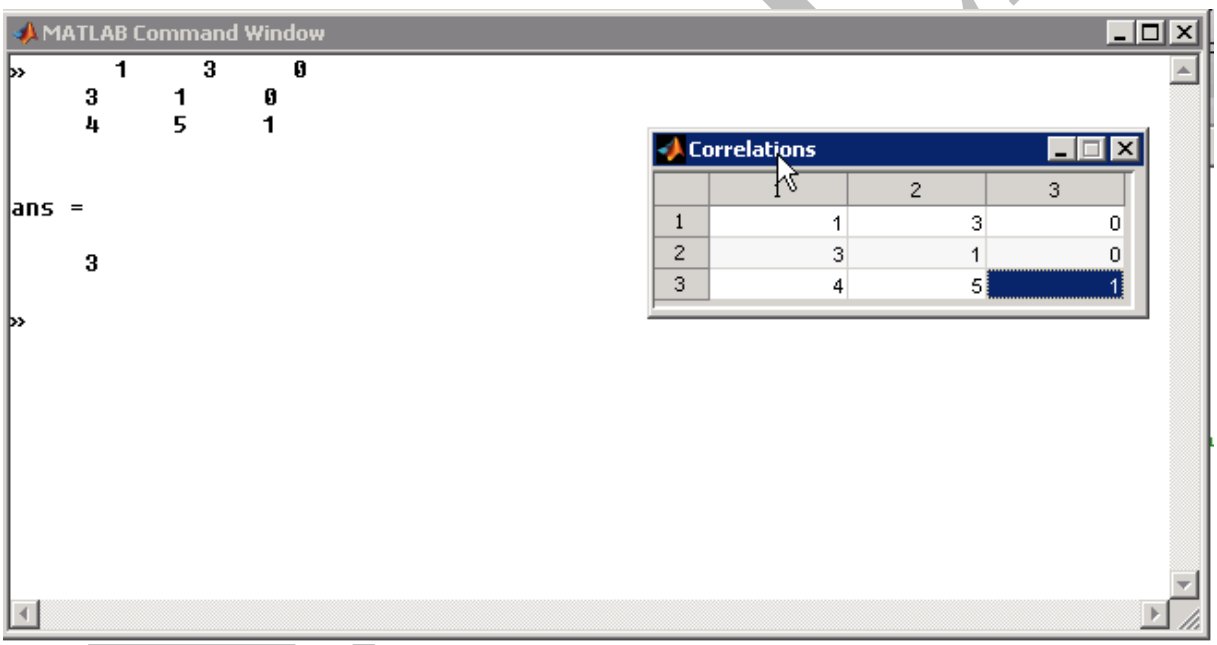


and we put:

Correlations.m*

```
76
77    % --- Executes when entered data in editable cell(s) in tbl_Correlation.
78    function tbl_Correlation_CellEditCallback(hObject, eventdata, handles)
79    % hObject    handle to tbl_Correlation (see GCBO)
80    % eventdata  structure with the following fields (see UITABLE)
81    %   Indices: row and column indices of the cell(s) edited
82    %   PreviousData: previous data for the cell(s) edited
83    %   EditData: string(s) entered by the user
84    %   NewData: EditData or its converted form set on the Data property. Empty if Data was not changed
85    %   Error: error string when failed to convert EditData to appropriate value for Data
86    % handles    structure with handles and user data (see GUIDATA)
87 -  data=get(hObject,'data');
88 -  disp(data); %display the whole matrix in the Command Window
89 -  data(1,2) %show only the first line, second columdn component
```

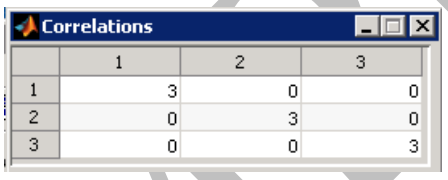
There when we chang the values of the table in the useform we get immediately the inpout in the Command Window:



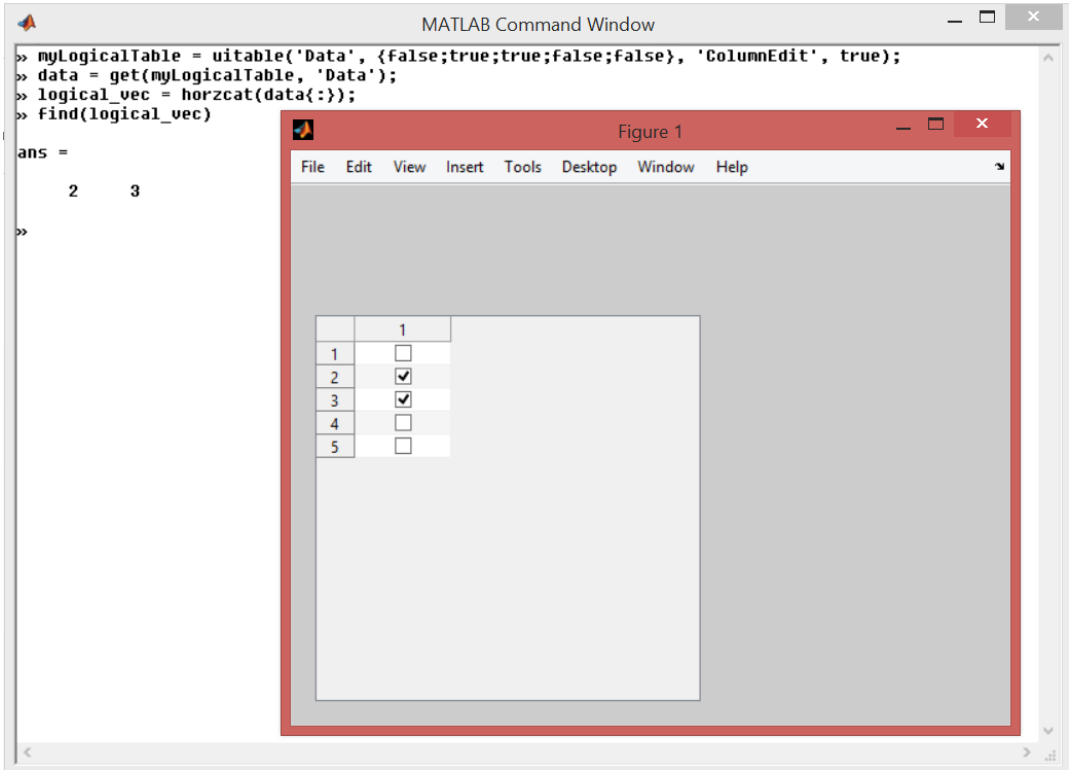
Now in the let us found the callback **_OpeningFcn** to add in it a specific matrix when the form loads:

```
Correlations.m
47    % --- Executes just before Correlations is made visible.
48    function Correlations_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.
50    % hObject    handle to figure
51    % eventdata  reserved - to be defined in a future version of MATLAB
52    % handles    structure with handles and user data (see GUIDATA)
53    % varargin   command line arguments to Correlations (see VARARGIN)
54
55    % Choose default command line output for Correlations
56    handles.output = hObject;
57
58    % Update handles structure
59    guidata(hObject, handles);
60    set(handles.tbl_Correlation,'Data',[3,0,0;0,3,0;0,0,3])
61
```

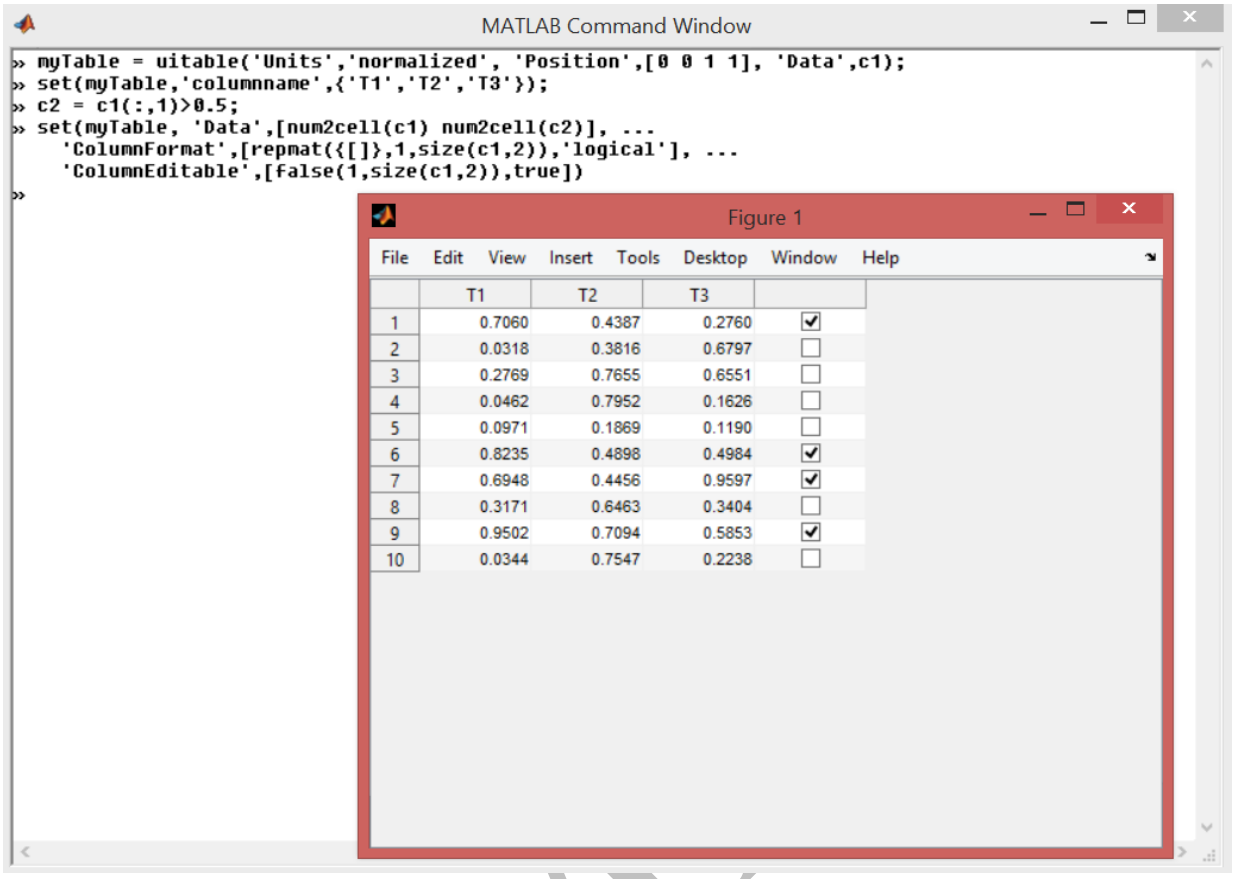
and as we can see it, this works well!:



There are other interesting stuffs about tables. For example adding checkboxes and retrieving the selected rows:

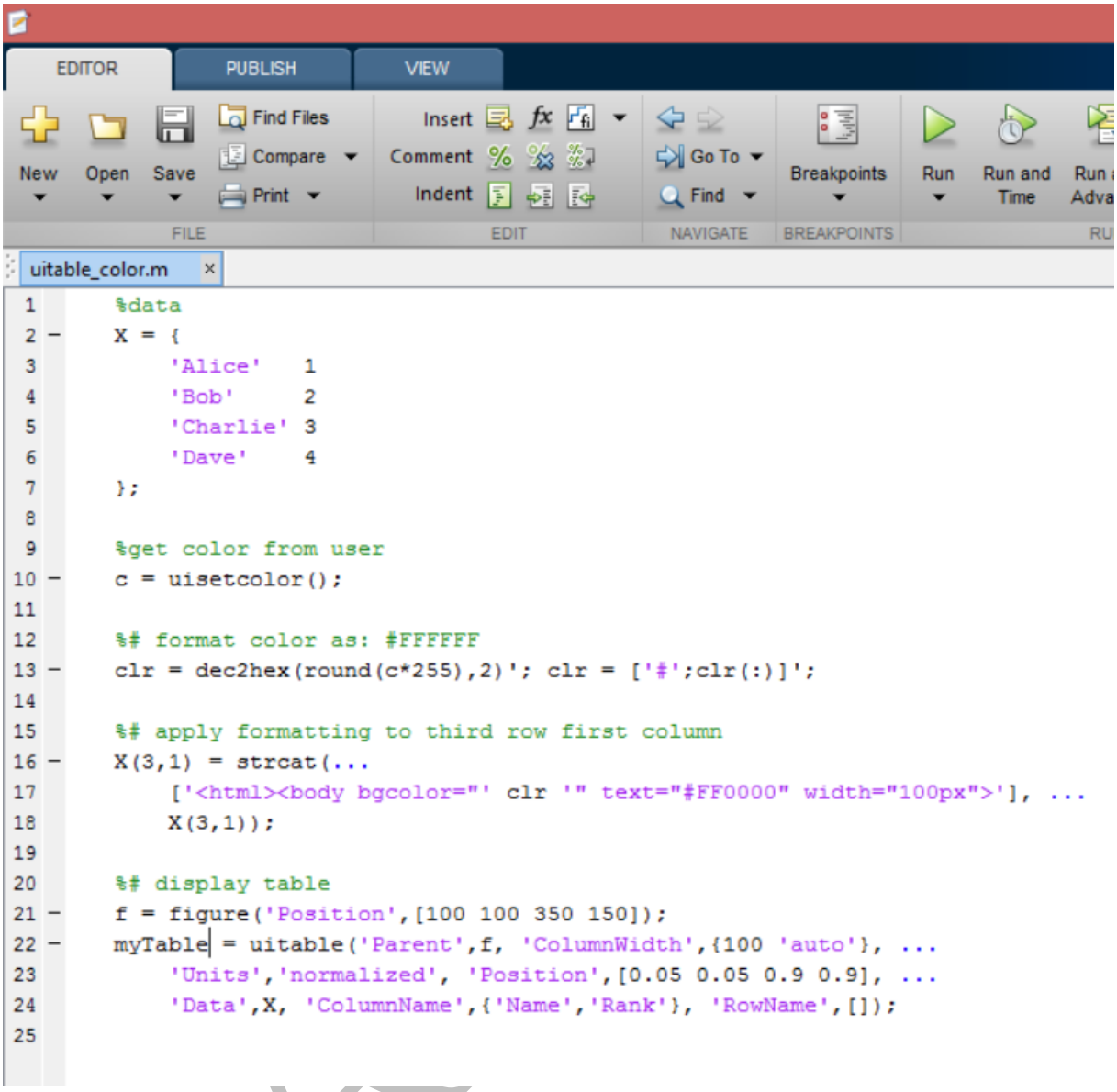


Or with multiple named columns and checkboxes:

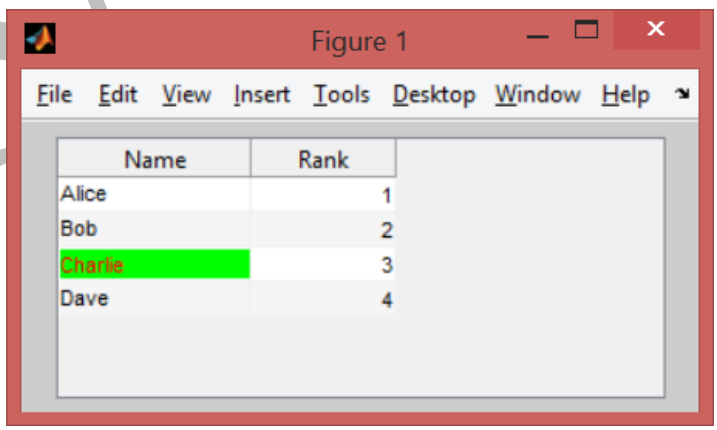


And what about colors in cells? Let us see an example:

SAMPLE

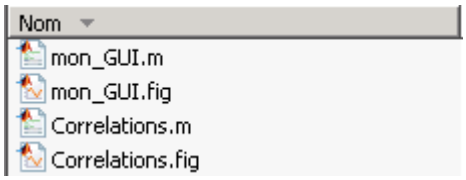

EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
Insert
Comment
Indent
Go To
Find
Breakpoints
Run
Run and Time
FILE
EDIT
NAVIGATE
BREAKPOINTS
uitable_color.m
%data
X = {
'Alice' 1
'Bob' 2
'Charlie' 3
'Dave' 4
};
%get color from user
c = uisetcolor();
%# format color as: #FFFFFF
clr = dec2hex(round(c*255),2)'; clr = ['#';clr(:)]';
%# apply formatting to third row first column
X(3,1) = strcat(...
['<html><body bgcolor="' clr '" text="#FF0000" width="100px">'], ...
X(3,1));
%# display table
f = figure('Position',[100 100 350 150]);
myTable = uitable('Parent',f, 'ColumnWidth',{100 'auto'}, ...
'Units','normalized', 'Position',[0.05 0.05 0.9 0.9], ...
'Data',X, 'ColumnName',{'Name','Rank'}, 'RowName',[]);

Then we get:

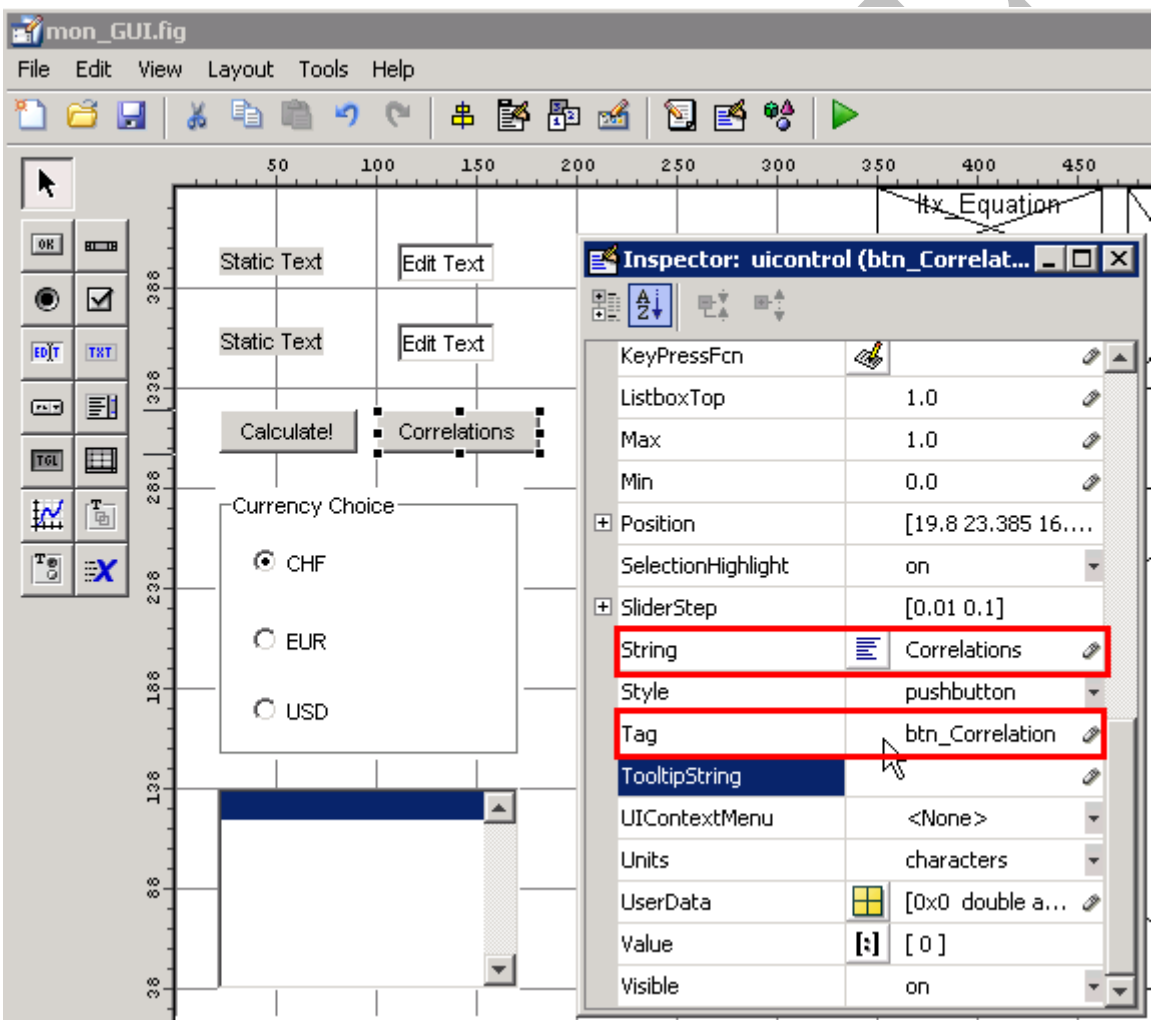

Figure 1
File Edit View Insert Tools Desktop Window Help
Name
Rank
Alice 1
Bob 2
Charlie 3
Dave 4

Userform calling another userform

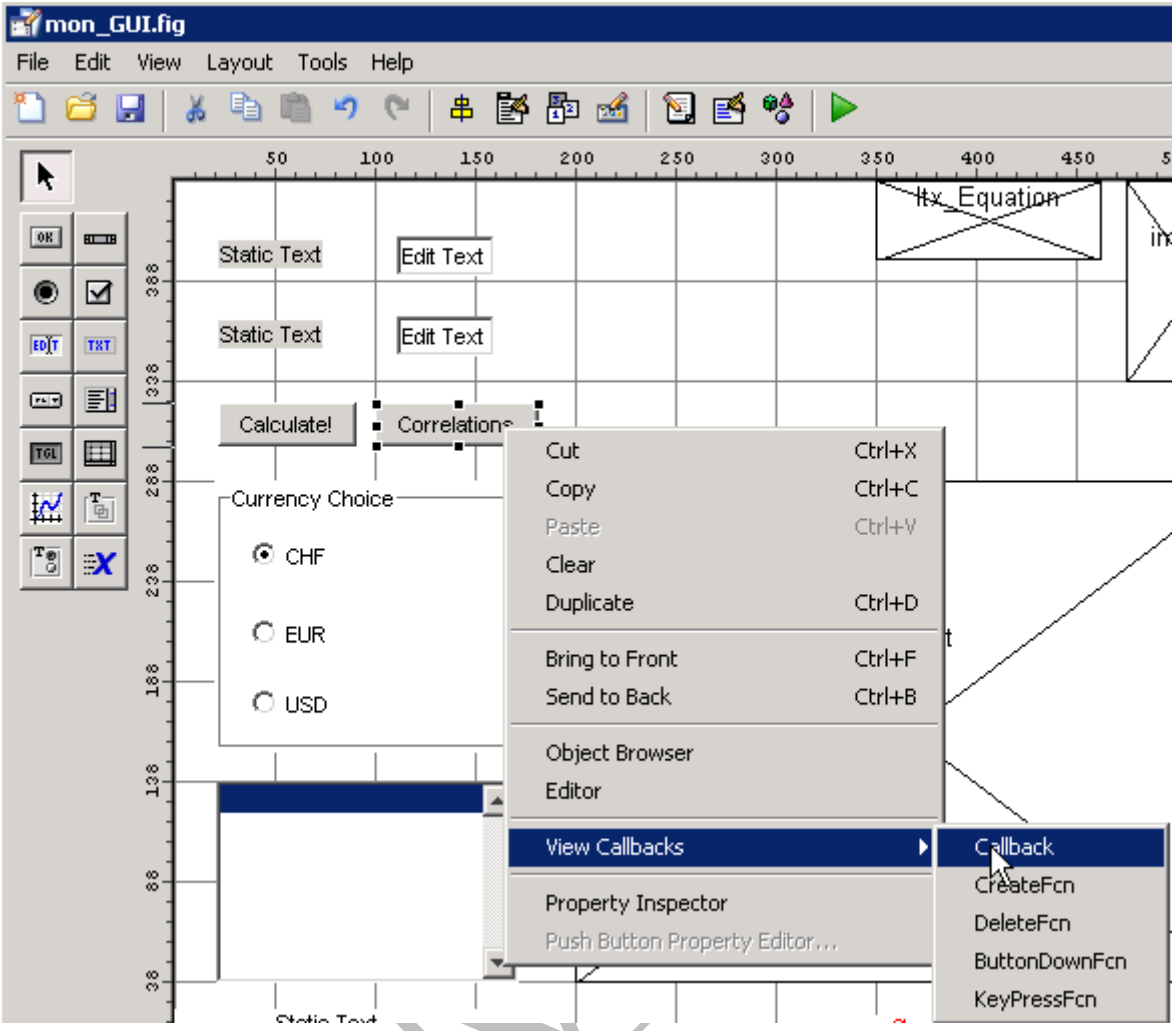
Let us see now hot to call a userform from another ruserform. Let us consider that we have a folder with the following file:



then we create a button to open *Correlations.fig* from *mon_GUI.fig* and we assign a caption and Tag to it:



as we know it will require a callback:



Where we simple add:

```
mon_GUI.m*
288    % --- Executes on button press in btn_Correlation.
289    function btn_Correlation_Callback hObject, eventdata, handles)
290    % hObject    handle to btn_Correlation (see GCBO)
291    % eventdata  reserved - to be defined in a future version of MATLAB
292    % handles    structure with handles and user data (see GUIDATA)
293    Correlations %the name of your "subform"
294
```

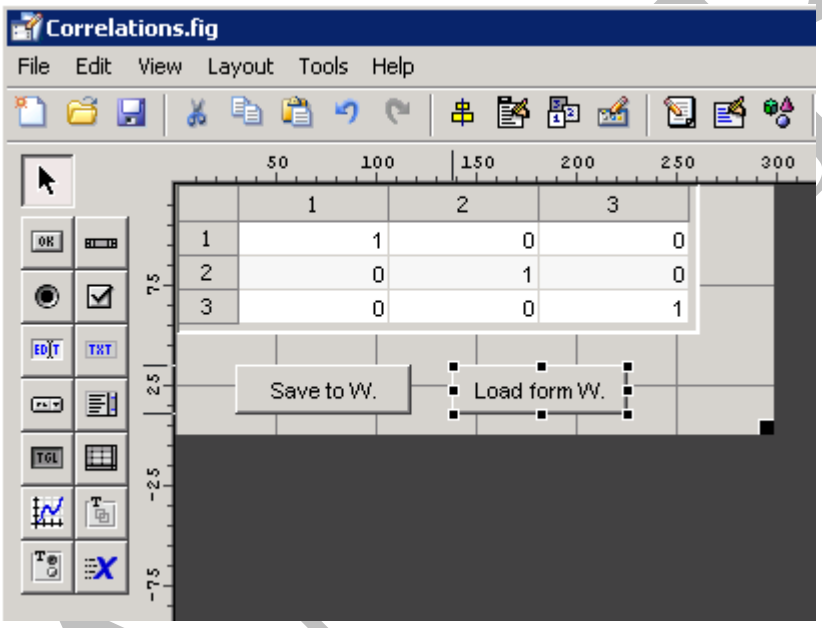
And that's it! You can now test you button and enjoy the result!

Write/Get GUI variables in MATLAB workspace

There exist methods to pass variables between GUI[2] without writing the variables in the workspace but I prefer this last solution for multiple reasons:

1. The user can change the variable on the fly from the *Command Window* if he prefers
2. The user can save all variables in a MAT-file to reload them later or for other use
3. The script is less horrible than the one

Suppose we want to be able to save or load our correlation matrix to the Workspace. First create two buttons *btn_SaveW* and *btn_LoadW* as following:

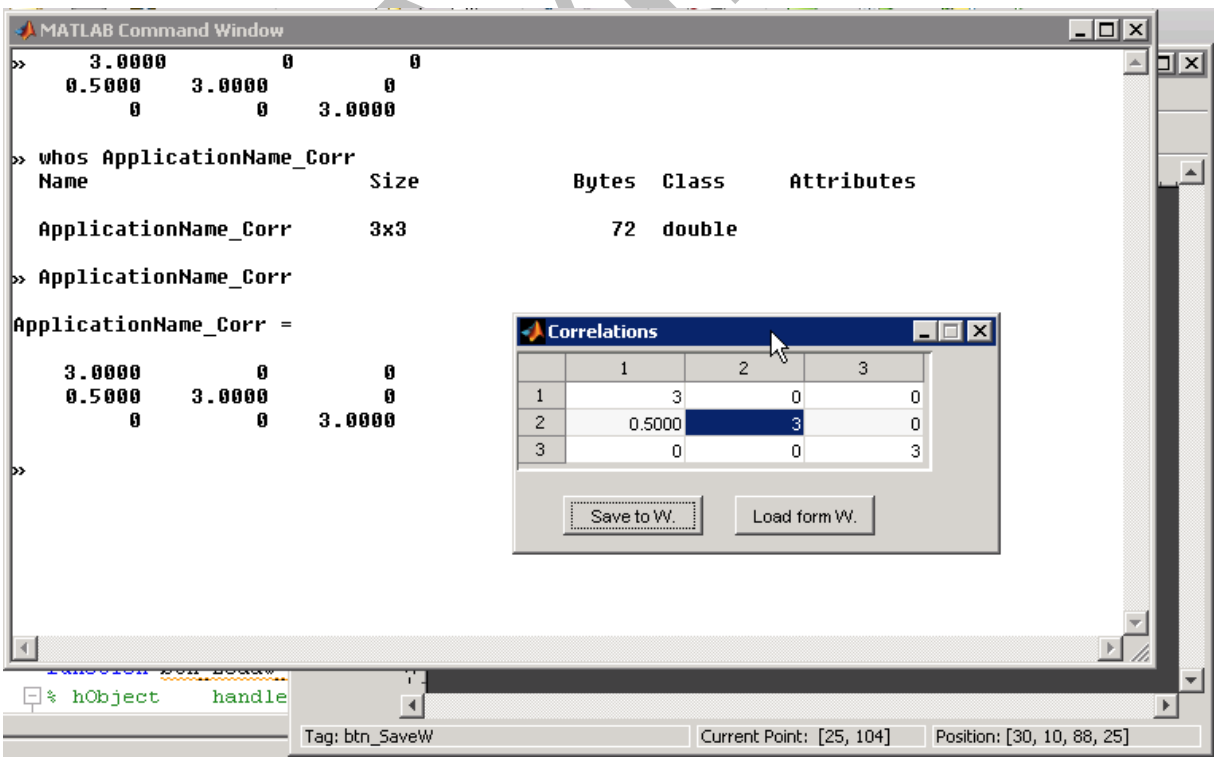


[2] To pass variables between Callbacks in the same GUI just add *global myvariable_name;* at the beginning of each callback where you need to access your variable and add at the end of the callback that fills you variables the command *guidata(hObject,handles)* to maintain the variable in the memory (see the example in the text).

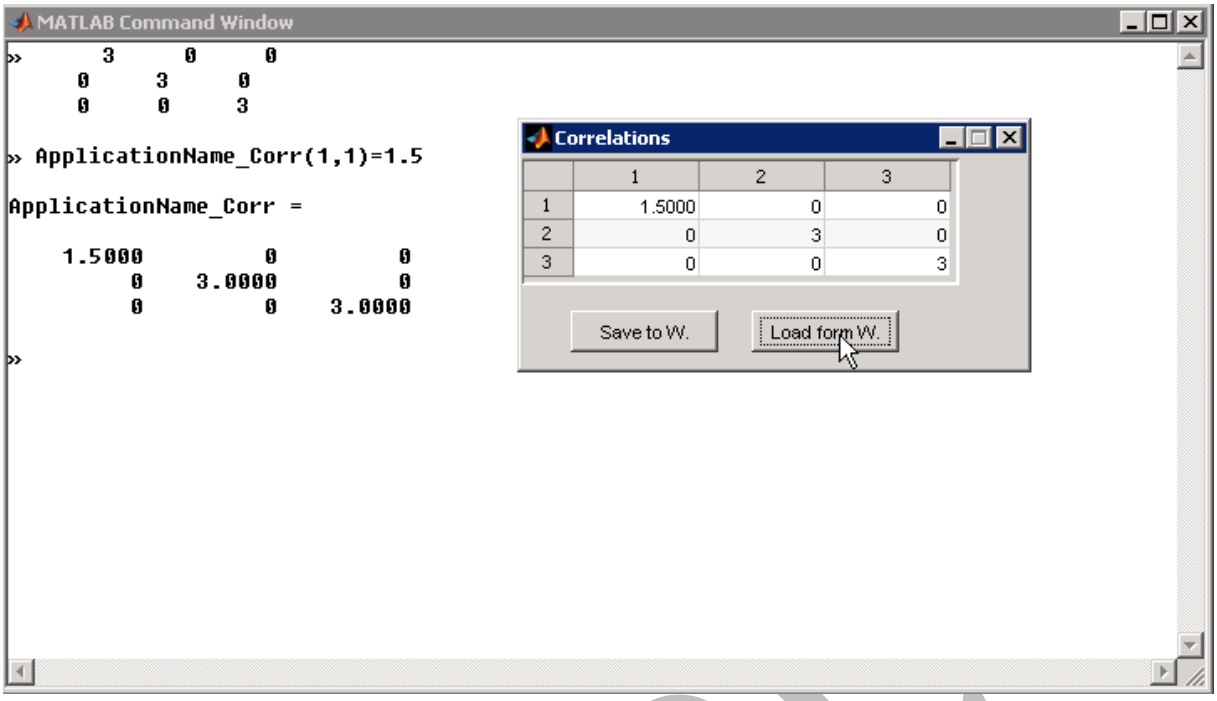
After change the all **Callbacks** as following using global variable and especially use the functions **assignin( )** and **evalin( )** to write and read in the workspace:

```
Correlations.m
 76
 77      % --- Executes when entered data in editable cell(s) in tbl_Correlation.
 78      function tbl_Correlation_CellEditCallback(hObject, eventdata, handles)
 79      % hObject    handle to tbl_Correlation (see GCBO) %...%
 87 -      global data;
 88 -      data=get(hObject,'data');
 89 -      guidata(hObject,handles);
 90
 91
 92      % --- Executes on button press in btn_SaveW.
 93      function btn_SaveW_Callback(hObject, eventdata, handles)
 94      % hObject    handle to btn_SaveW (see GCBO) %...%
 97 -      global data;
 98 -      assignin('base','ApplicationName_Corr',data);
 99 -      disp(data)
100
101      % --- Executes on button press in btn_LoadW.
102      function btn_LoadW_Callback(hObject, eventdata, handles)
103      % hObject    handle to btn_LoadW (see GCBO) %...%
106 -      set(handles.tbl_Correlation,'Data',evalin('base','ApplicationName_Corr'))
107
```

When you click on the button *Save to W.* you can see and control in the **Command Window** that everything is working:



And also you can change the variable in the workspace and after click on the button *Load from W.* to see that everything works well:


MATLAB Command Window
» 3 0 0
0 3 0
0 0 3
» ApplicationName_Corr(1,1)=1.5
ApplicationName_Corr =
1.5000 0 0
0 3.0000 0
0 0 3.0000
»
Correlations
1 2 3
1 1.5000 0 0
2 0 3 0
3 0 0 3
Save to W.
Load form W.

SAMPLE

Create userforms using only script

Some people don't like, or simply cannot, create GUI using the MATLAB WYSWIG builder. Therefore it can be useful to see how to crate GUI only with pure script (very useful if you don't have MATLAB on your computer).

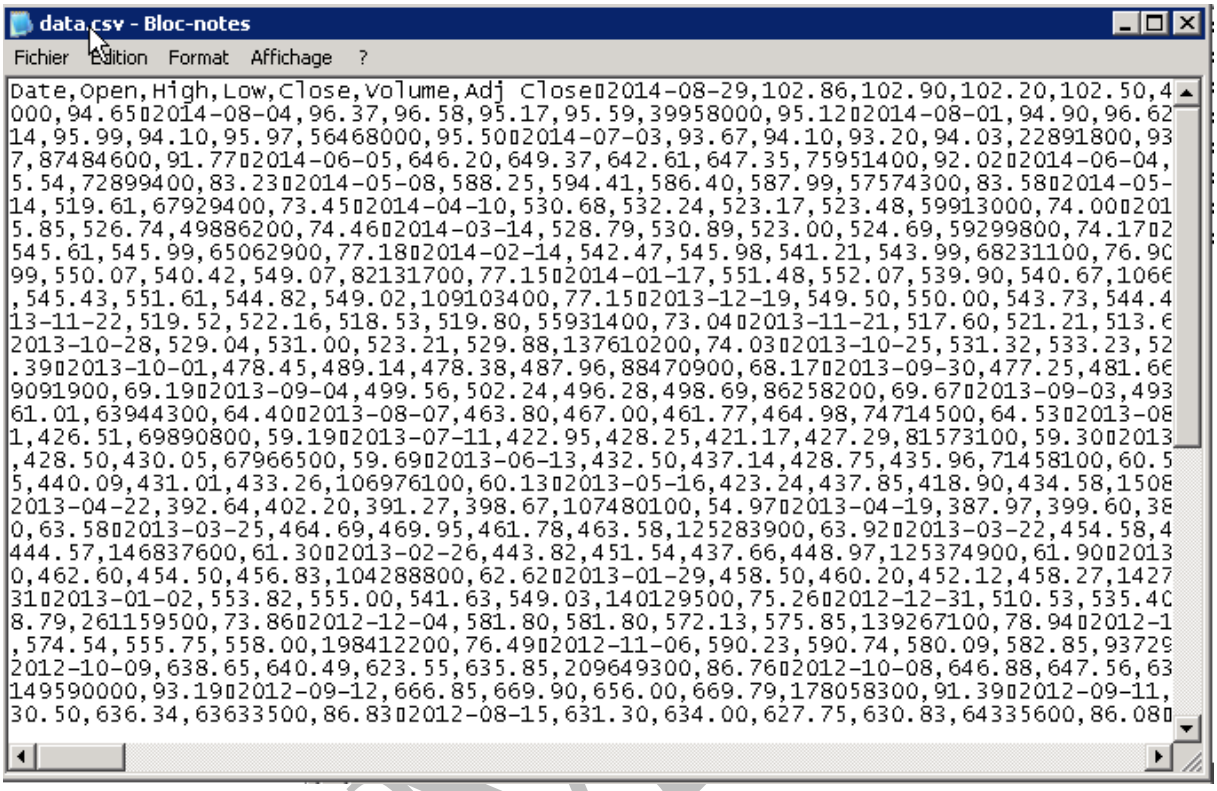
ToDo…

SAMPLE

Important practical cases

Create a GUI that opens a file

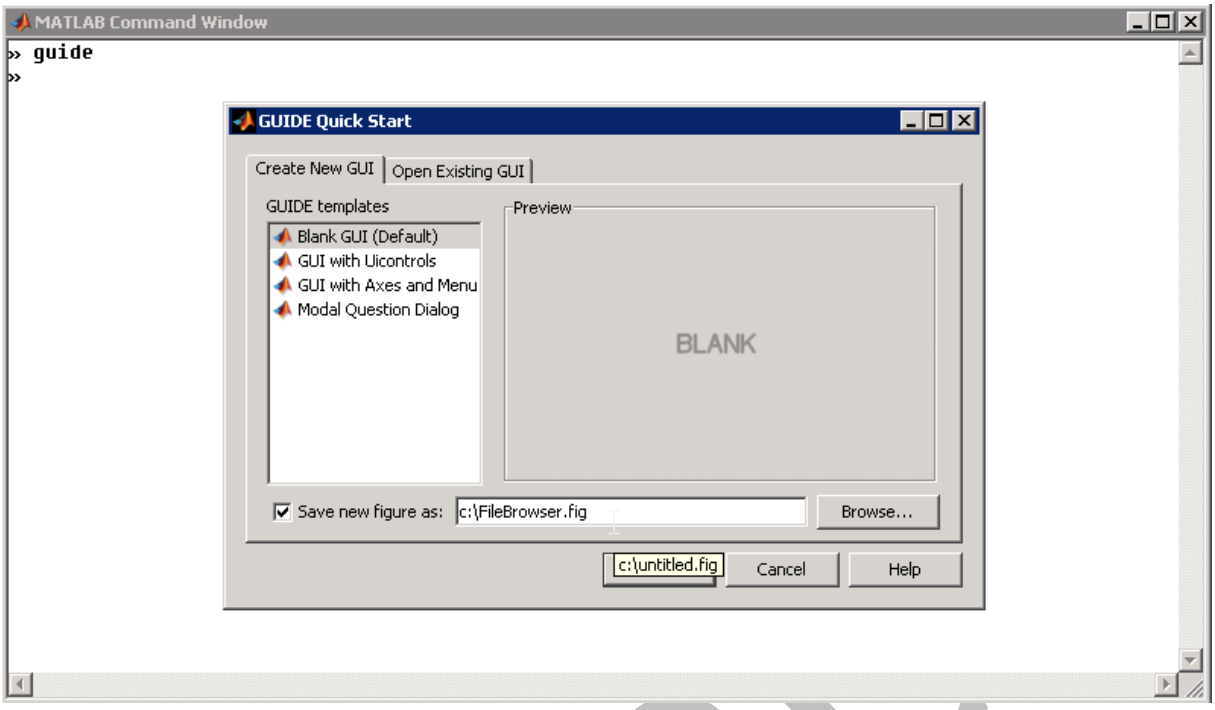
The idea will be to see how it looks like to load such a file:



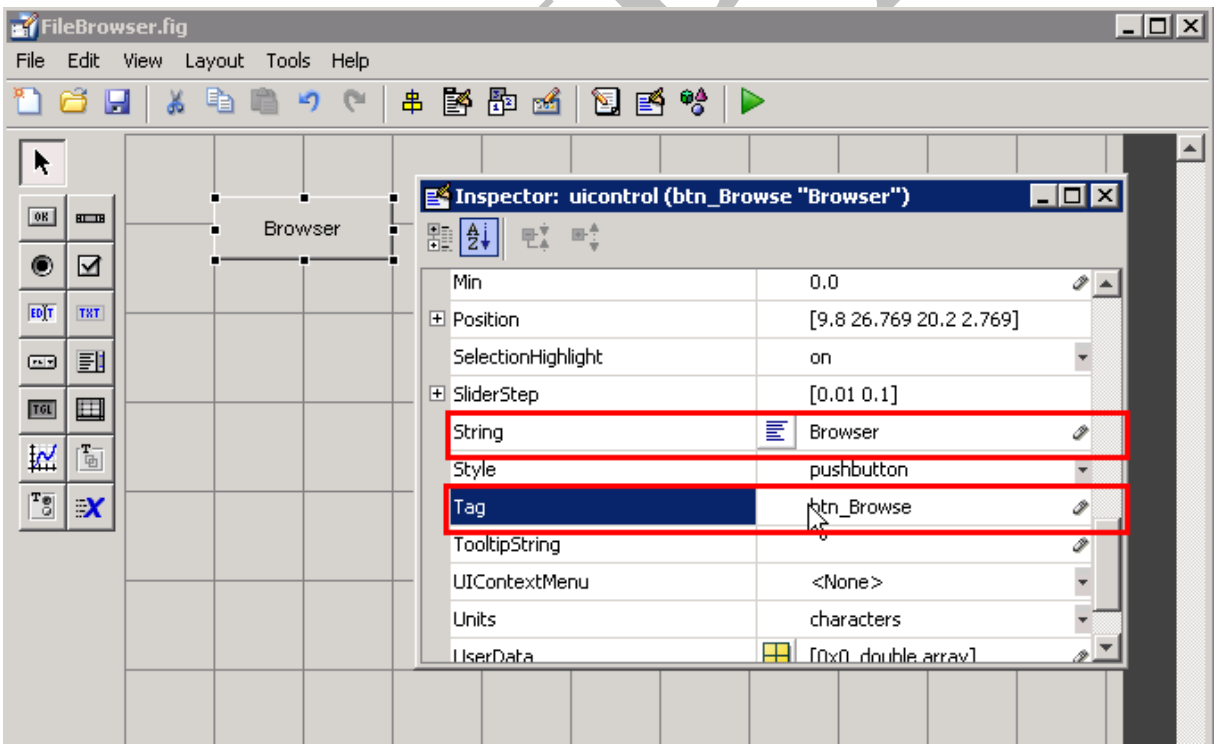
in a MATLAB GUI **Static text** area.

First we create the GUI:

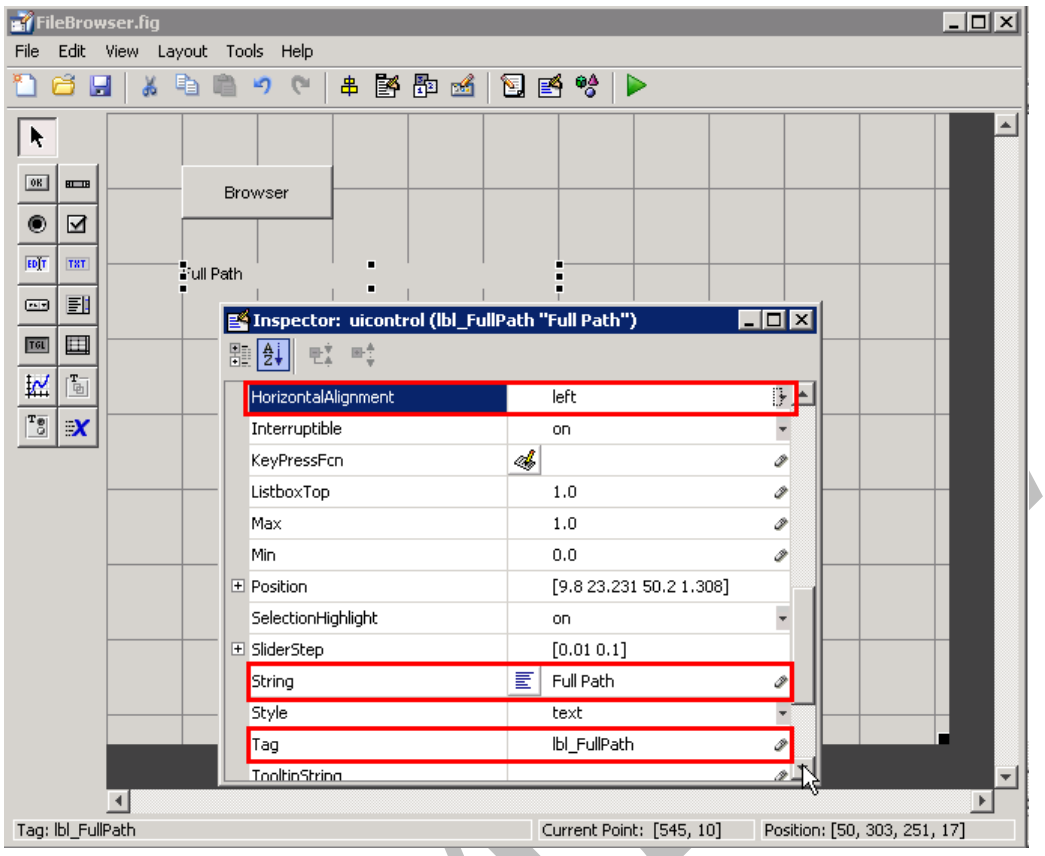
SAM



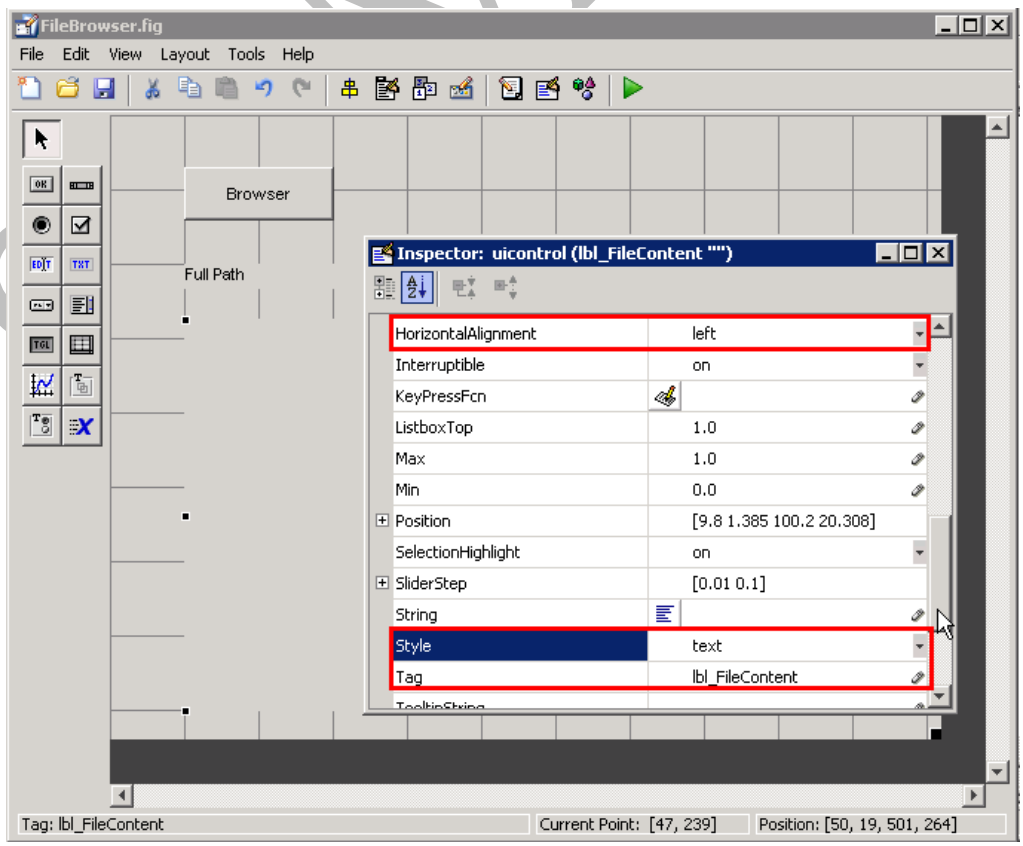
We change the **String** property and the **Tag**:



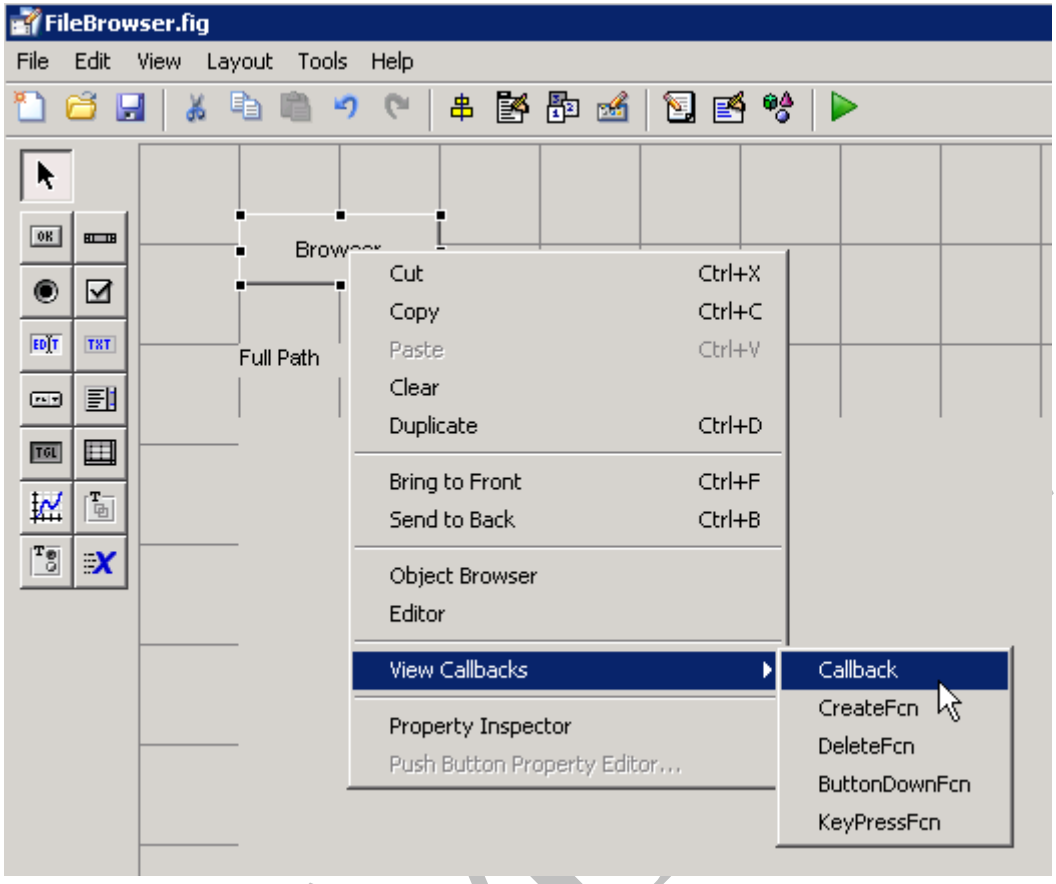
We add a **Static text** area and change a few standard stuff:



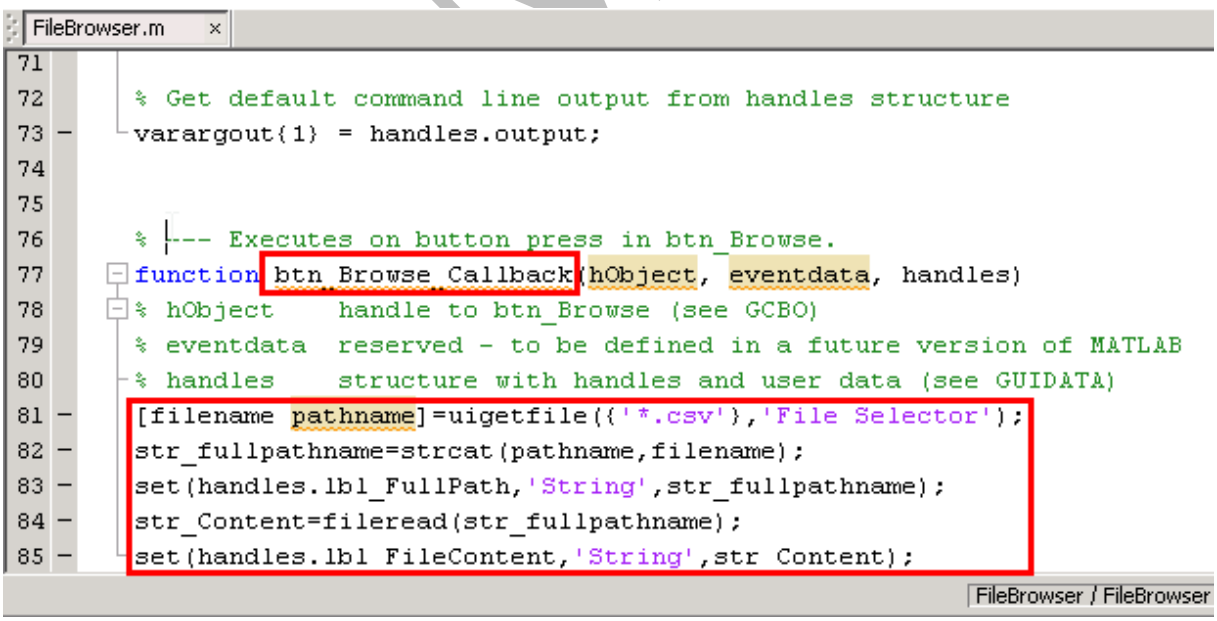
We add the main Static text area afterwards:



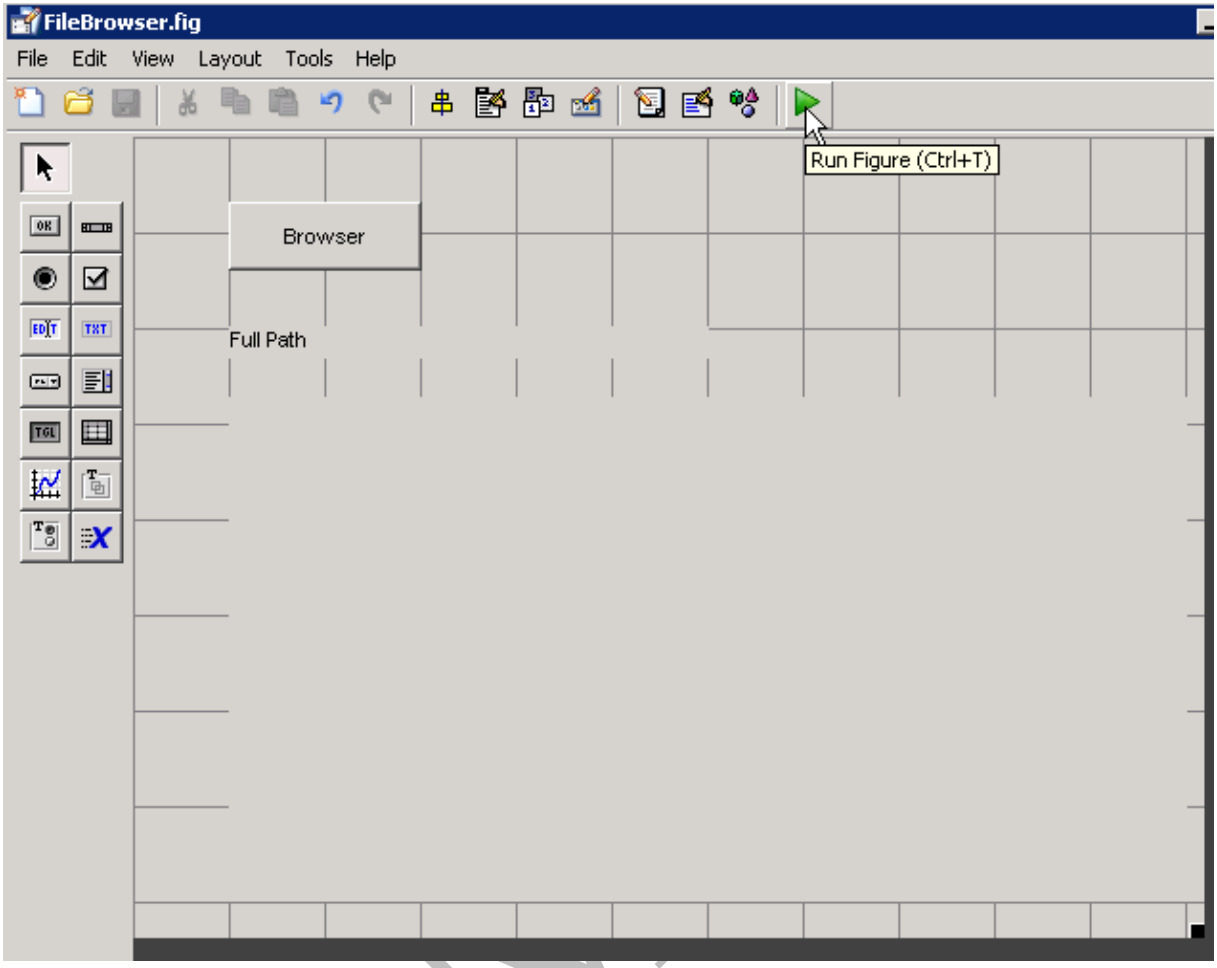
Now we click make the **Callback** script of the main button visible:



And we add the following:

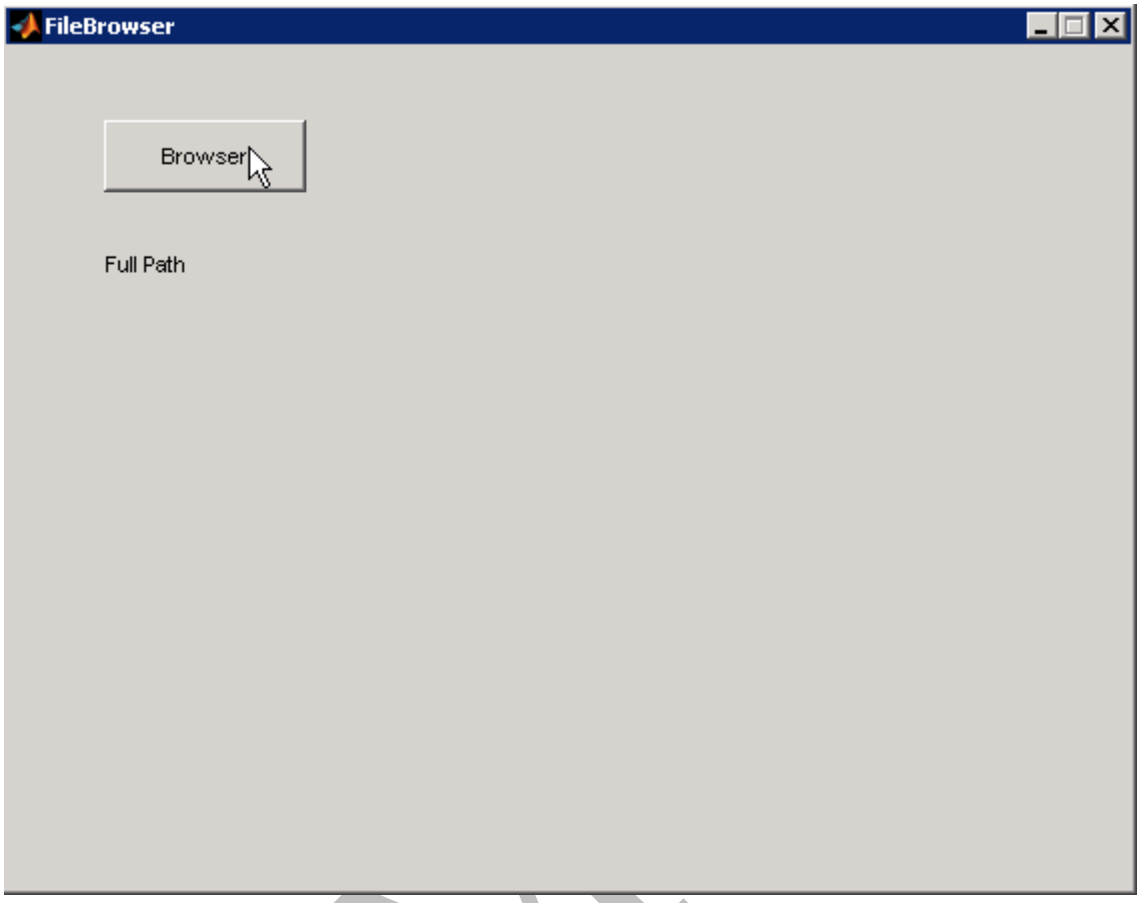


Now we can test it:

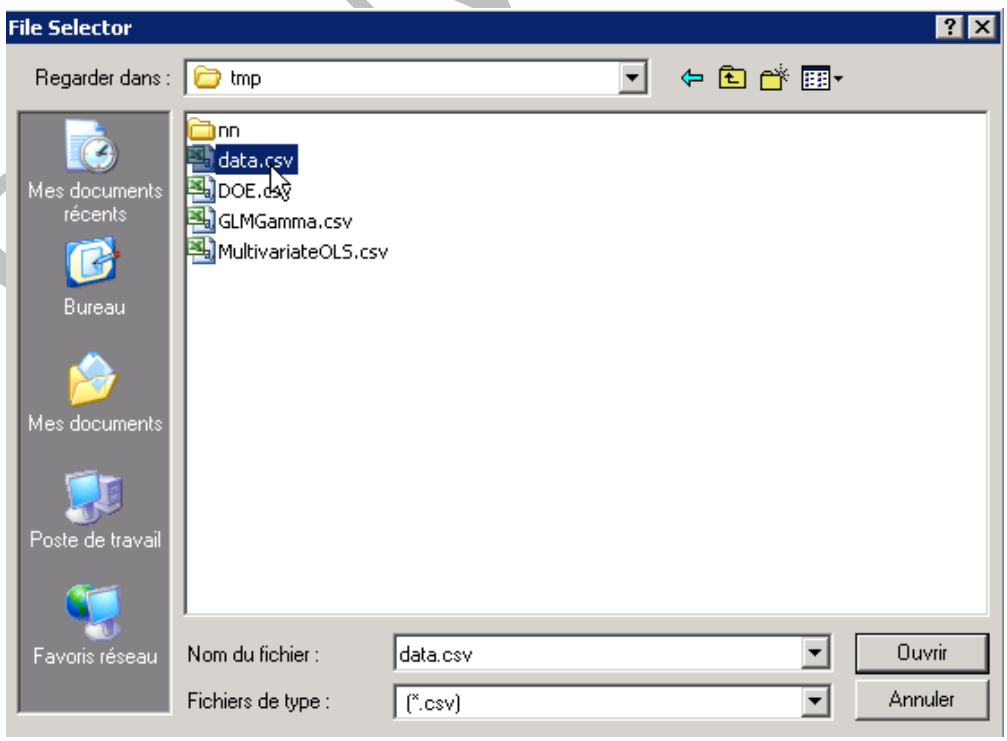


SAM

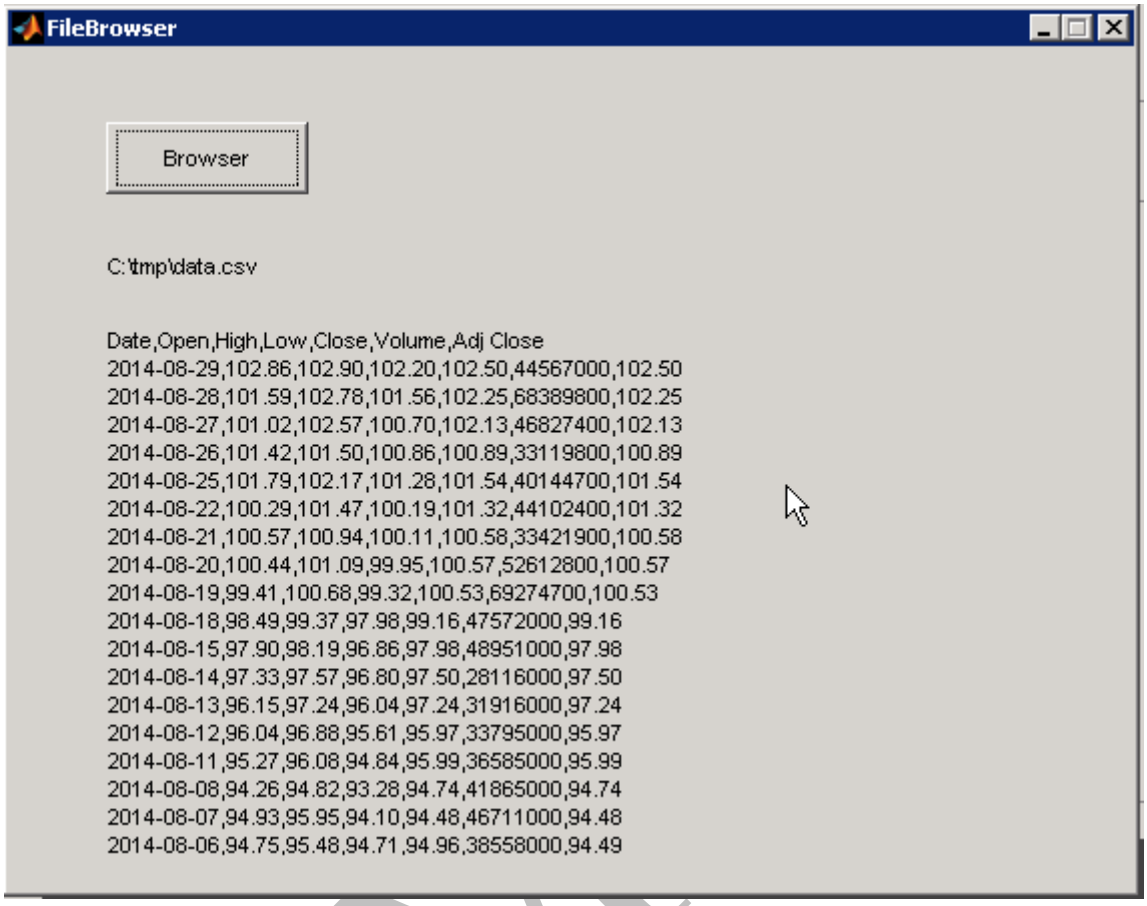
Once executed it looks like this:



When we click on *Browse* we get as expected:



Once the *.csv file choose we have the result we wanted:



SAM

Object Oriented Programming (OOP)

This section is particularly important for people that code 3D applications or advanced softwares based on MATLAB framework.

First remember some definitions:

- **Class:** template for creating *objects*, defining properties and methodes, as weel as default values/behavior (e.g.: Car)
- **Object:** instance of a *class* that has a state (*properties*) and behavior (*methods*) (e.g.: Mercedes)
- **Properties:** data associated with an object (e.g.: number of wheels).
- **Methods:** functions (behavior) defined in a class and associated with an object (e.g.: start car).
- **Attributes:** modify behavior of classes and class components (e.g.: car color)
- **Events:** start when something happens (e.g.: engine warm up when you turn key)
- **Listener:** code waiting for something
- **Inheritance:** object or class (subclass) derived from another object or class (superclass)
- **Polymorphism:** single interface to entities of different types

MVC (Model-Vue-Controller)

We normally don't need to present this development strategy mainly used for statefull applications strongly inspired from the ANSI/SPARC database concepts.

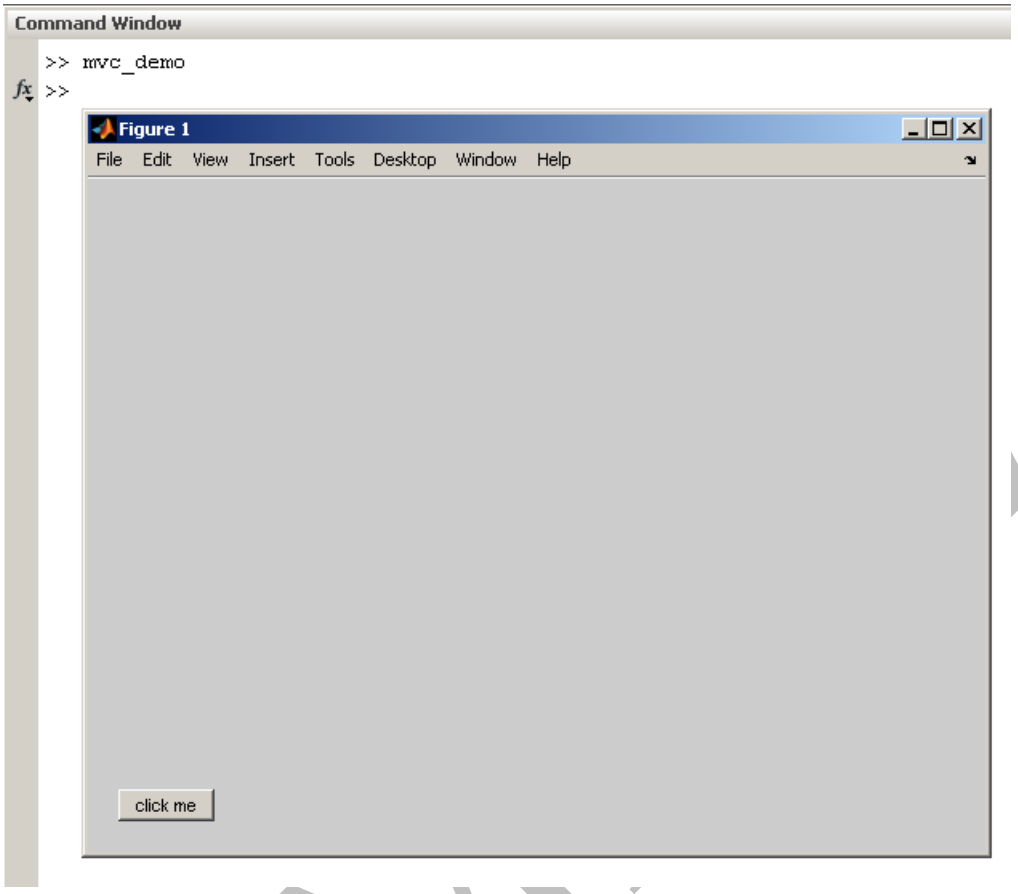
Here is a simple example:

mvc_demo.m

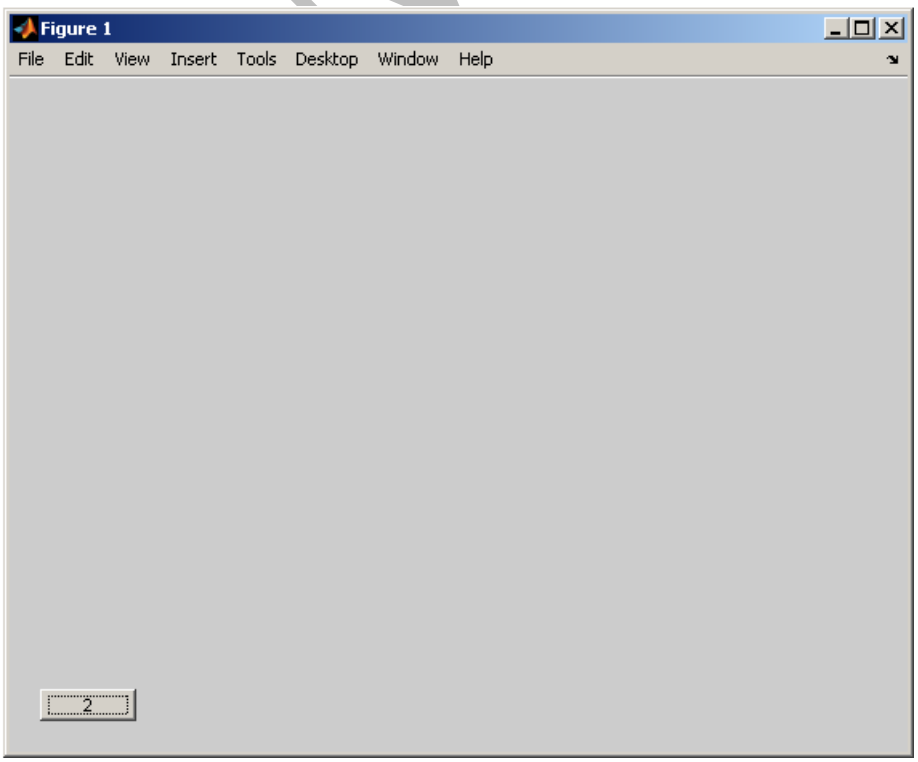
```
function mvc_demo()
    %THIS is nested scope variable
     this.model = getModel();
     this.view = getView();
     %create model
     function model = getModel()
         model.data1 = 1;
     end
     %create view
     function view = getView()
         view.hfig = figure();
         view.hbtn = uicontrol( 'style', 'push', 'string', 'click me', 'callback', @btn1_cb );
     end
     %controller
     function btn1_cb(varargin)
         this.model.data1 = this.model.data1 + 1;
         set( this.view.hbtn, 'string', num2str(this.model.data1) );
     end
end
```

SAMPLE

If we run it we get:



And if we click on the button we get:



Symbolic calculations (Symbolic Toolbox)

Now that we have finished with the core basic functions of MATLAB, it is time to focus on the first Toolbox in order of importance that is the: Symbolic Toolbox.

You will see in this chapter the most common commands use in business and engineering and also discover the similarities between MATLAB and Maple about the subjects covered in my book of Applied Mathematics.

New main core features (to be complete)

Note that all changes are not documented here! Please refer to the official release notes for a full list.

http://www.MathWorks.ch/ch/help/symbolic/release-notes.html

MATLAB 2014a (8.3)

- Quickly compute cosinus/sinus hyperbolic
- max and min for finding the largest and smallest elements of a symbolic array

MATLAB 2013b (8.2)

- *cumsum* and *cumprod* functions for computing cumulative sums and products
- close function to close MuPAD notebooks from MATLAB

MATLAB 2013a (8.1)

- Linear algebra functions for computing matrix factorizations (*lu*, *chol*), pseudoinverse, orthogonal basis, and adjoint
- *sign* function for computing signs of numbers

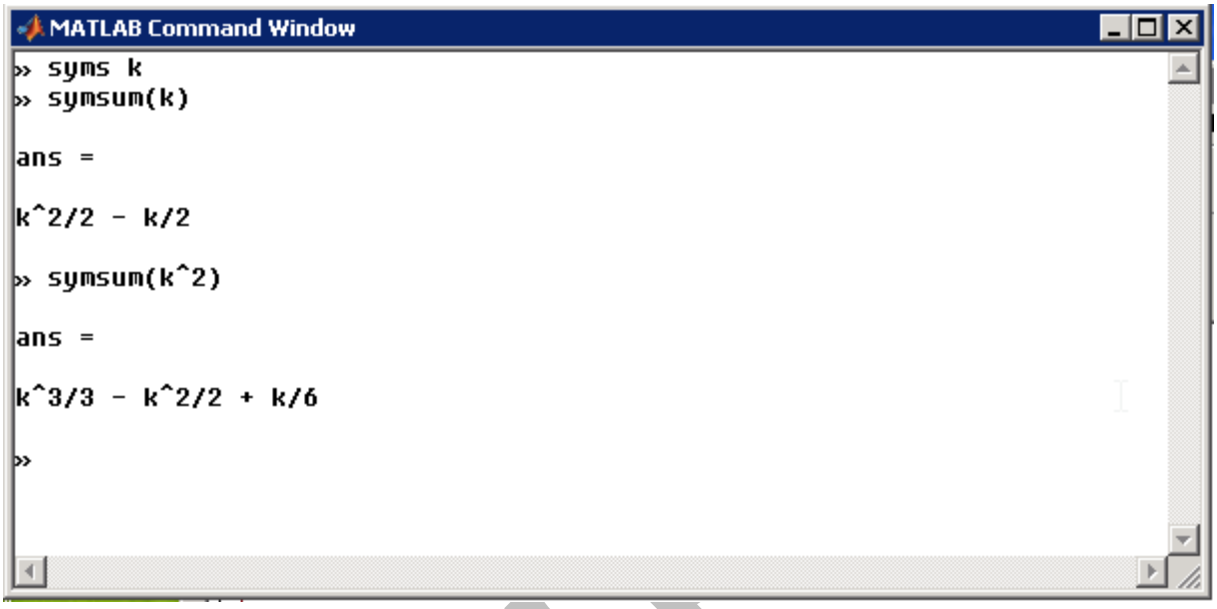
MATLAB 2012b (8.0)

- matrix analysis functions for characteristic *charpoly*
- *poles* function for determining the poles of an expression
- Functions for converting linear systems of equations to matrix form AX=B (*equationsToMatrix*) and solving matrix equations (*linsolve*)
- symbolic functions for describing pulses: rectangularPulse and triangularPulse

SAMPLE

Series

Have a look how MATLAB compute classics Gauss series as we proved them in the theoretical course and that are very useful in actuarial science We will use for this purpose the function **symsum( )**:



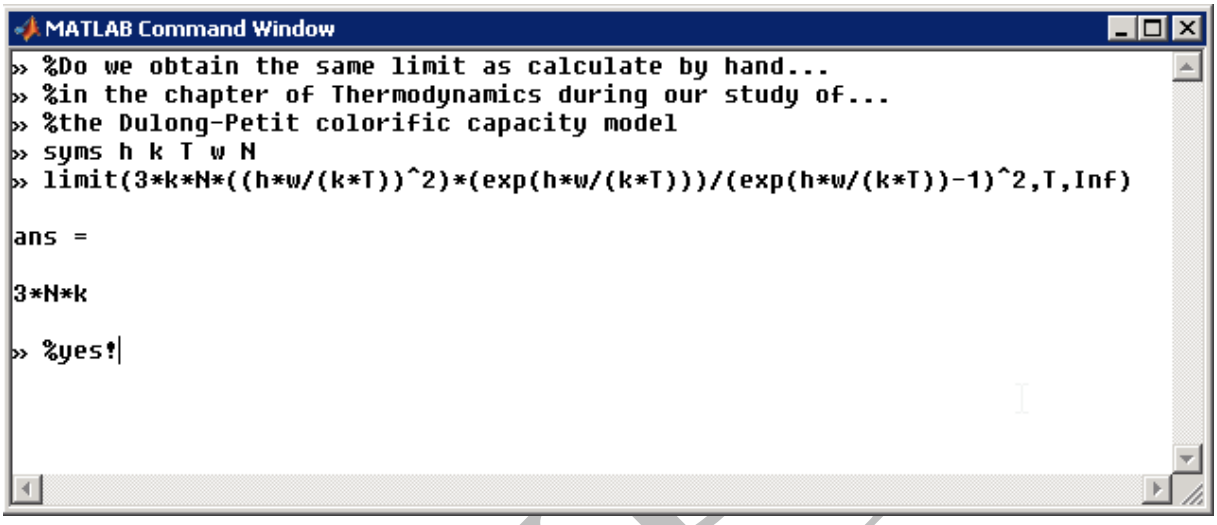
That's right. Just the esthetic of the result is missing!

SAMPLE

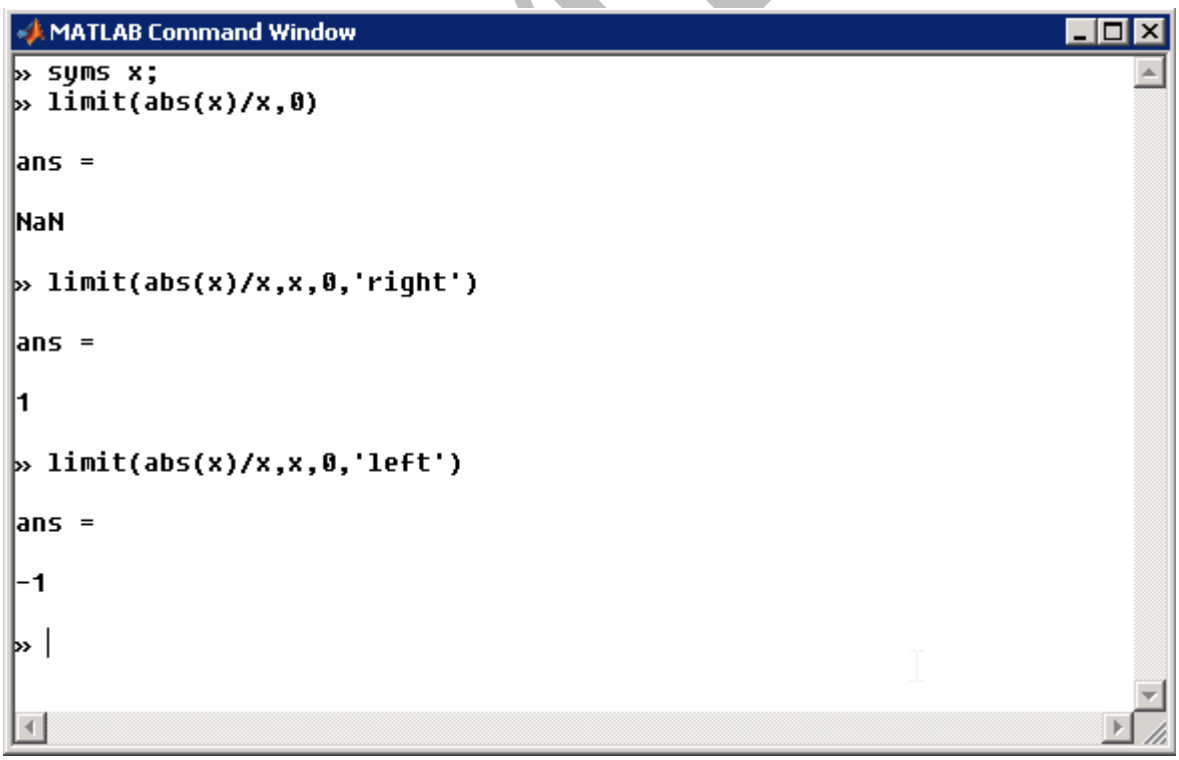
Limits

The first important used command and one of the most simple is the **limit( )** function to calculate the... limit O_o

For this purpose we will also need to learn how to create variable or constants symbols using the command **syms**:

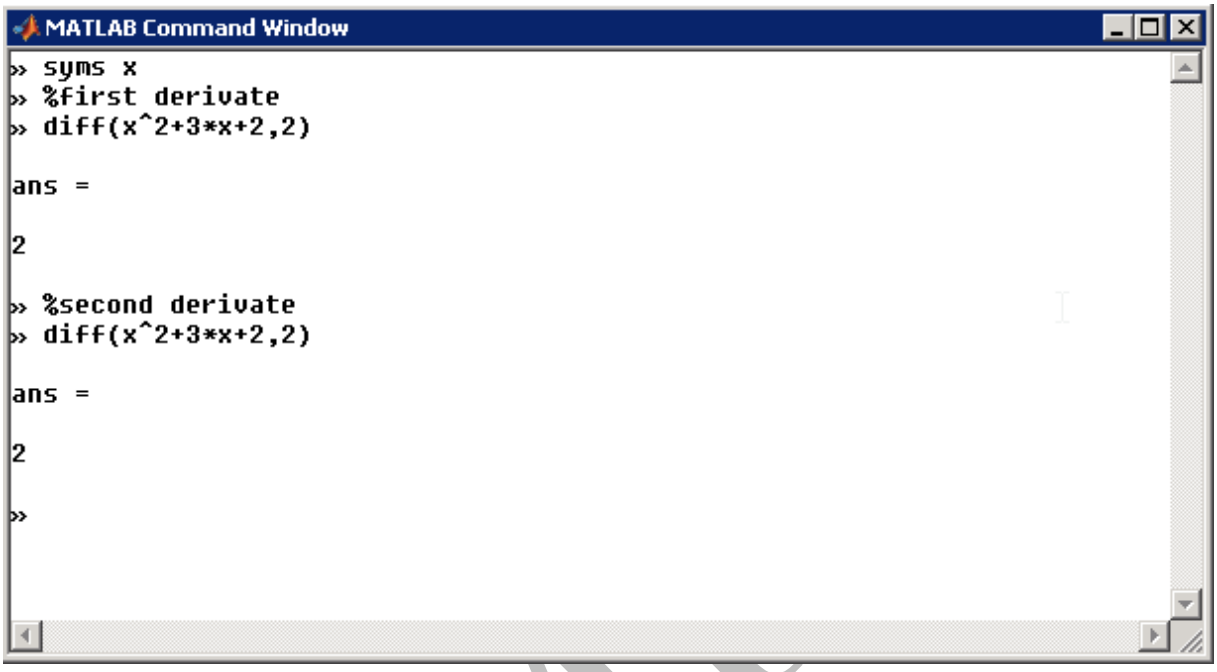


Or another well know case (trap) about limits:

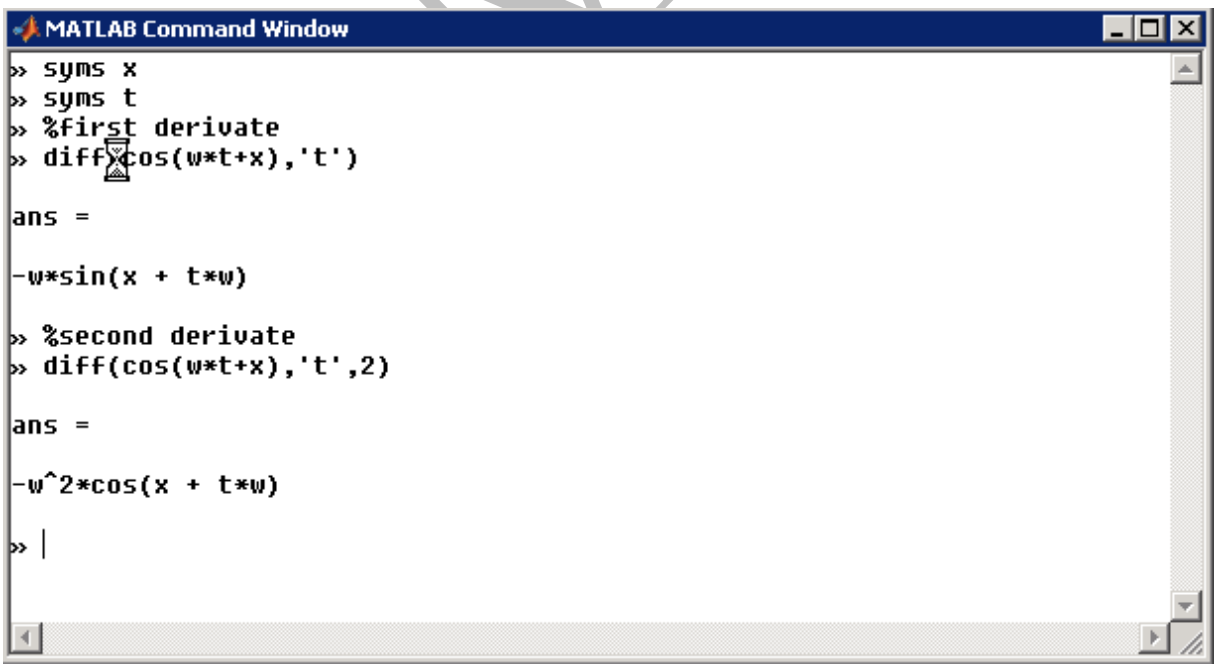


Derivate/Partial Derivate

As we save with the core version of MATLAB, derivation was limited to pseudo-symbolic polynomial only. With the Symbolic Toolbox we can derivate almost all classical functions and estimate their value on a given point using the function **diff( )**:

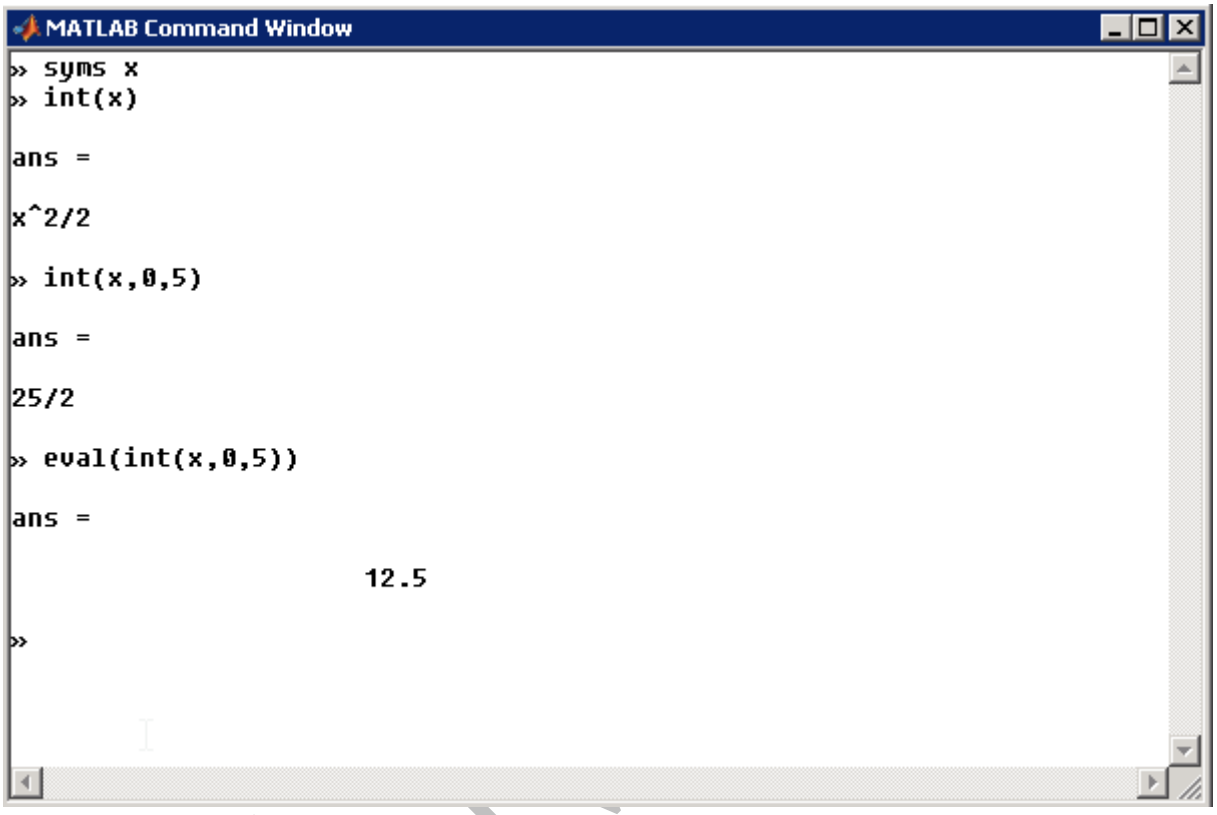


Or if we want a pure symbolic derivate without being interested to the evaluation on a given point we have (always using the **diff( )** function):



Integration

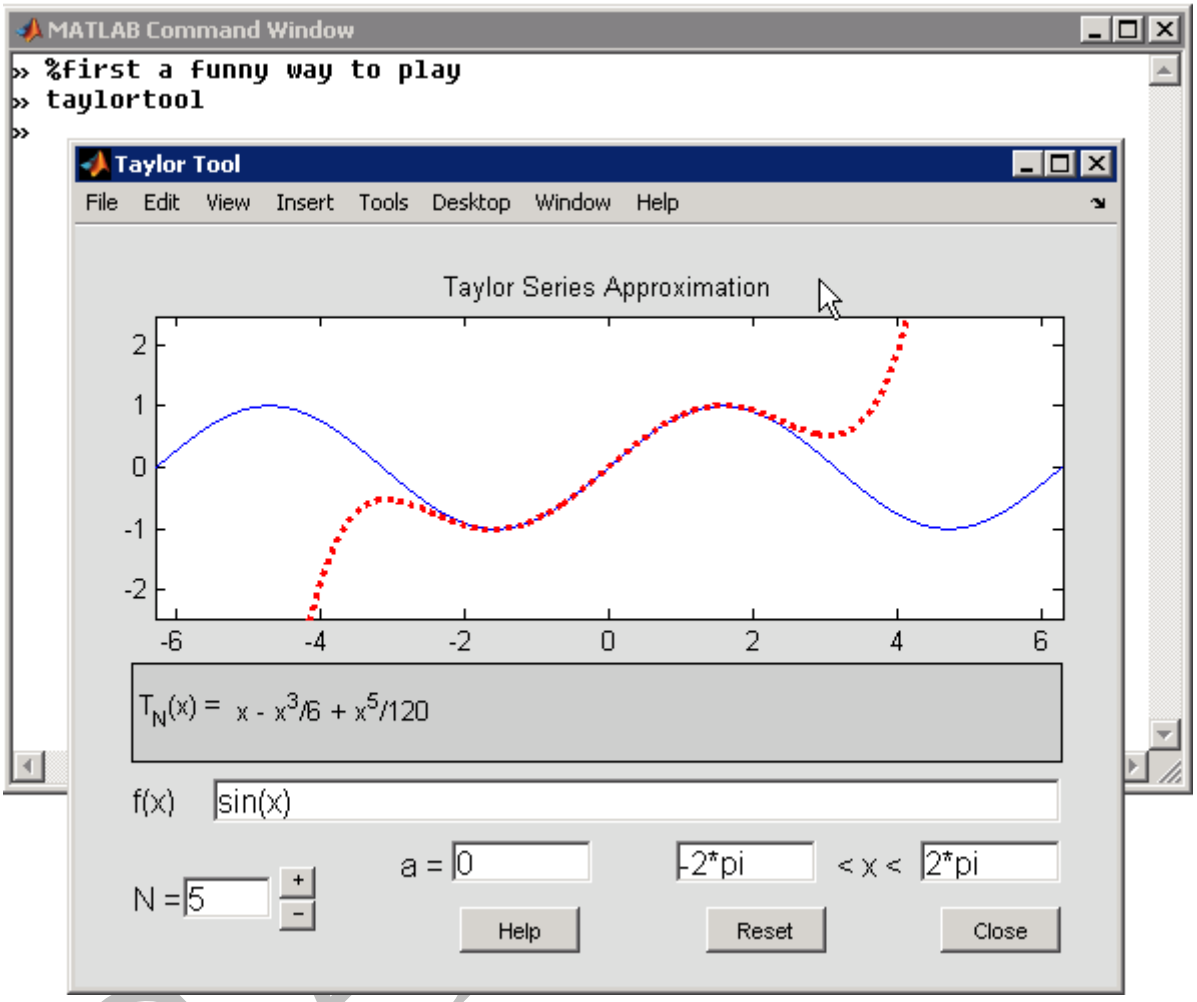
As we save with the core version of MATLAB, derivation was limited to pseudo-symbolic polynomial only. With the Symbolic Toolbox we can integrate almost all classical functions and estimate their value on a given point using the function **int( )** alone or associate with **eval( )**:



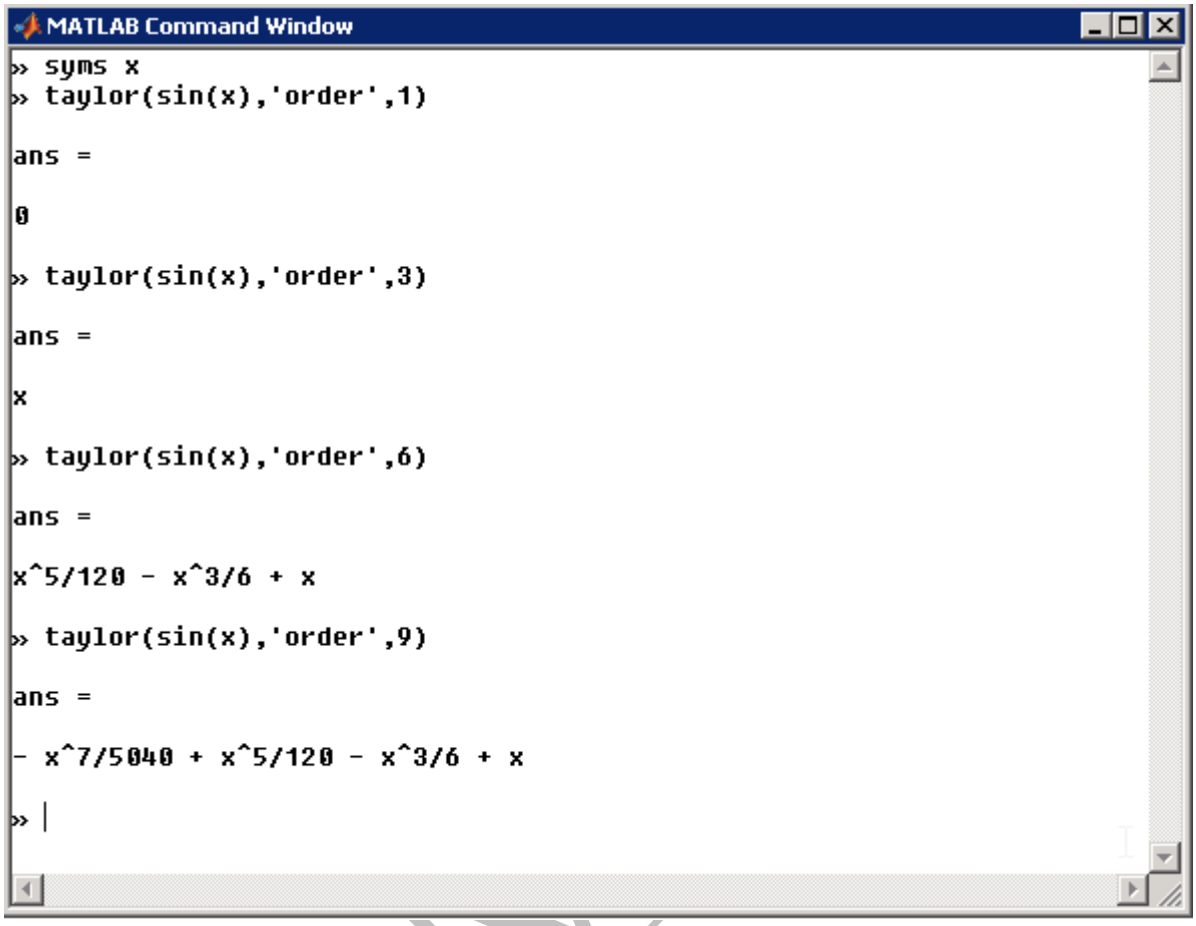
SAM

Taylor series

The fact that Taylor series expansions are not available in the core version of MATLAB is a huge and boring limitation. At least, it is available in the Symbolic Toolbox and there are two ways of using it. The first one is through a GUI wizard called trough the command **taylortool**:



The way to get directly in the *Command Windows* the Taylor expansion is by using the function **taylor( )** with some trivial arguments:



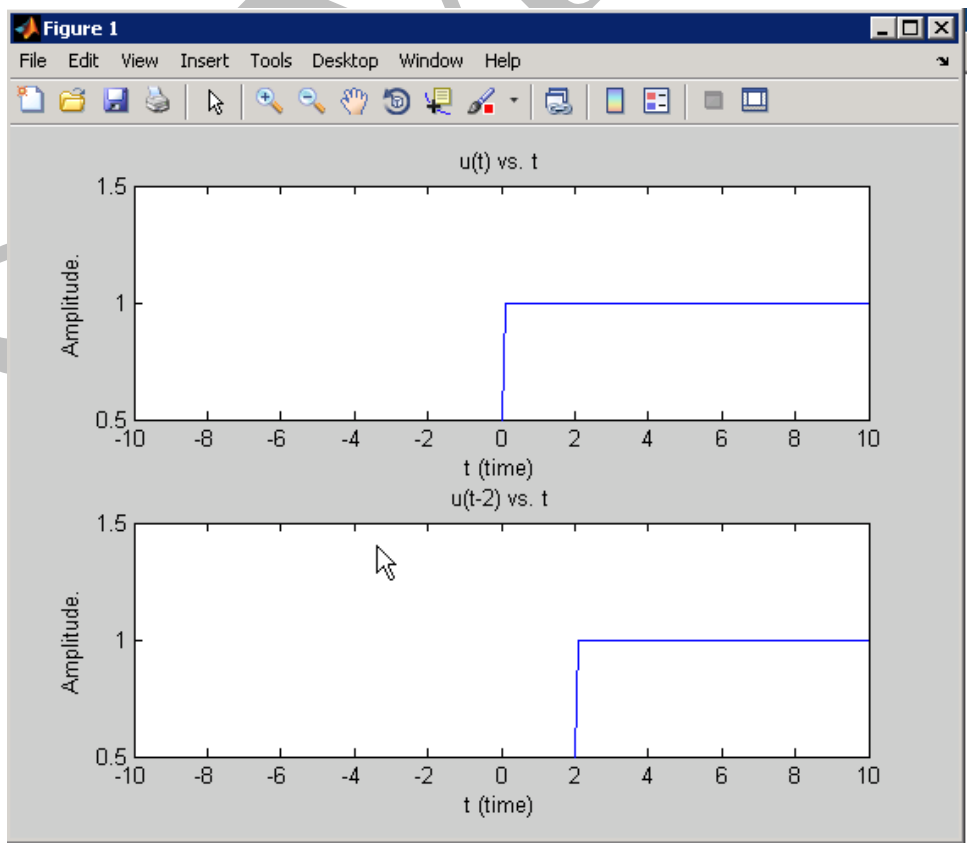
SAM

Heaviside function

If you have the symbolic toolbox you can use the **heaviside( )** function to get a step function (very useful in electronics):

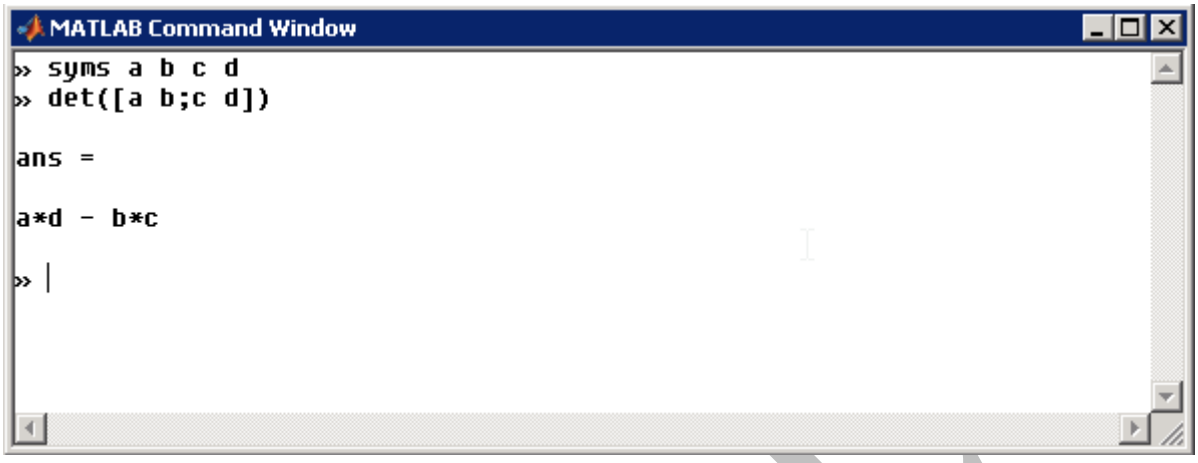
```
1   x = -10:0.1:10;
2   stepfun = heaviside(x);
3   subplot(2, 1, 1);
4   plot(x, stepfun)
5   xlabel('t (time)')
6   title('u(t) vs. t')
7   ylabel('Amplitude.')
8   axis([-10 10 ,0.5 1.5])
9   subplot(2, 1, 2);
10  stepfun = heaviside(x-2);
11  plot(x, stepfun)
12  axis([-10 10 ,0.5 1.5])
13  title('u(t-2) vs. t')
14  ylabel('Amplitude.');
15  xlabel('t (time)');
```

So we get:



Matrix determinant

We saw that in the core MATLAB version it was very easy to get the determinant of a numeric matrix with the function **det( )**. By passing to the same function variables instead of numerical values we can also get the determinant:



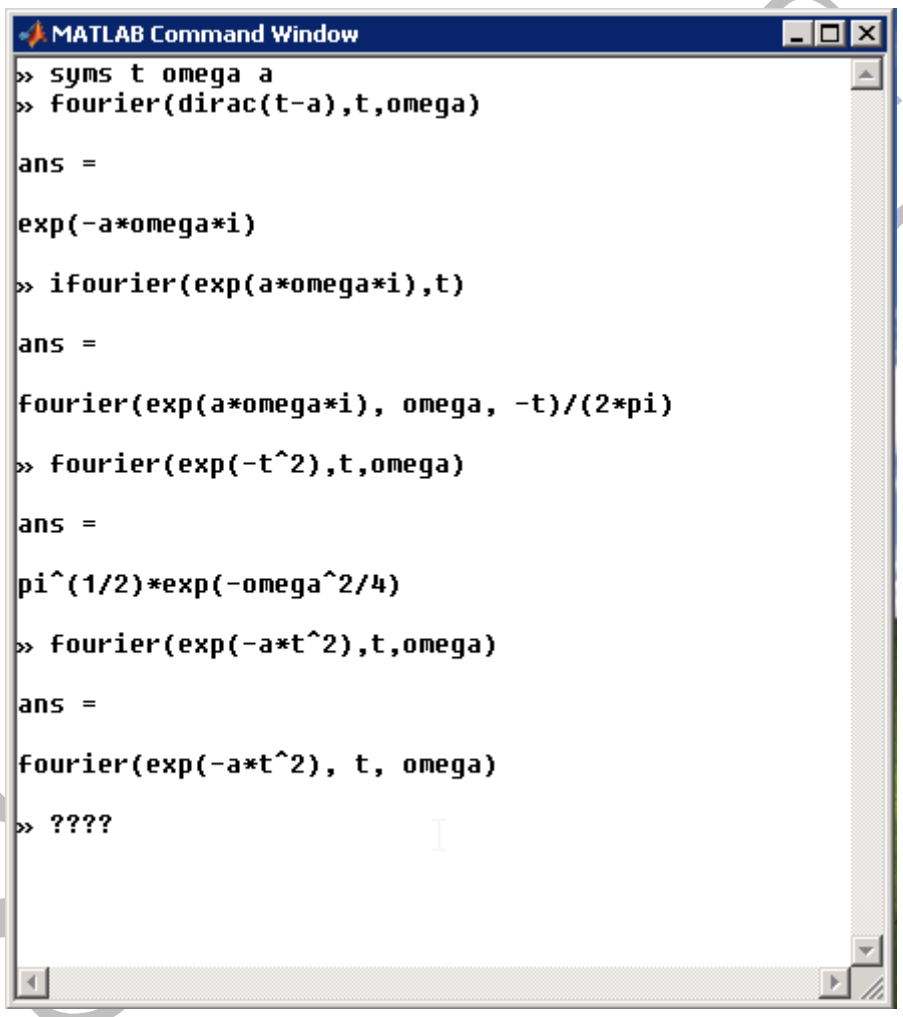
SAMPLE

Integral Transformations

In the theoretical course we have seen how powerful are Fourier, Laplace or Z transformations to solve differential equations, or for calculating series values or for signal processing. The purpose here is just to make an example of each to see that we obtain the same results as those calculated by hand.

Fourier Transform

Have a look to some Fourier transformation we calculated by hand in the theoretical course:



Everything seems ok except the last one (heat equation Fourier Transform) that MATLAB seems not be able to compute. But in reality as we know it, it is because of the sign of *a*. Thus:

```
MATLAB Command Window
» syms t omega
» syms a positive
» f=exp(-a*t^2);
» fourier(f,t,omega)

ans =

(pi^(1/2)*exp(-omega^2/(4*a)))/a^(1/2)

»
```

We have also the famous result useful for resolving PDE:

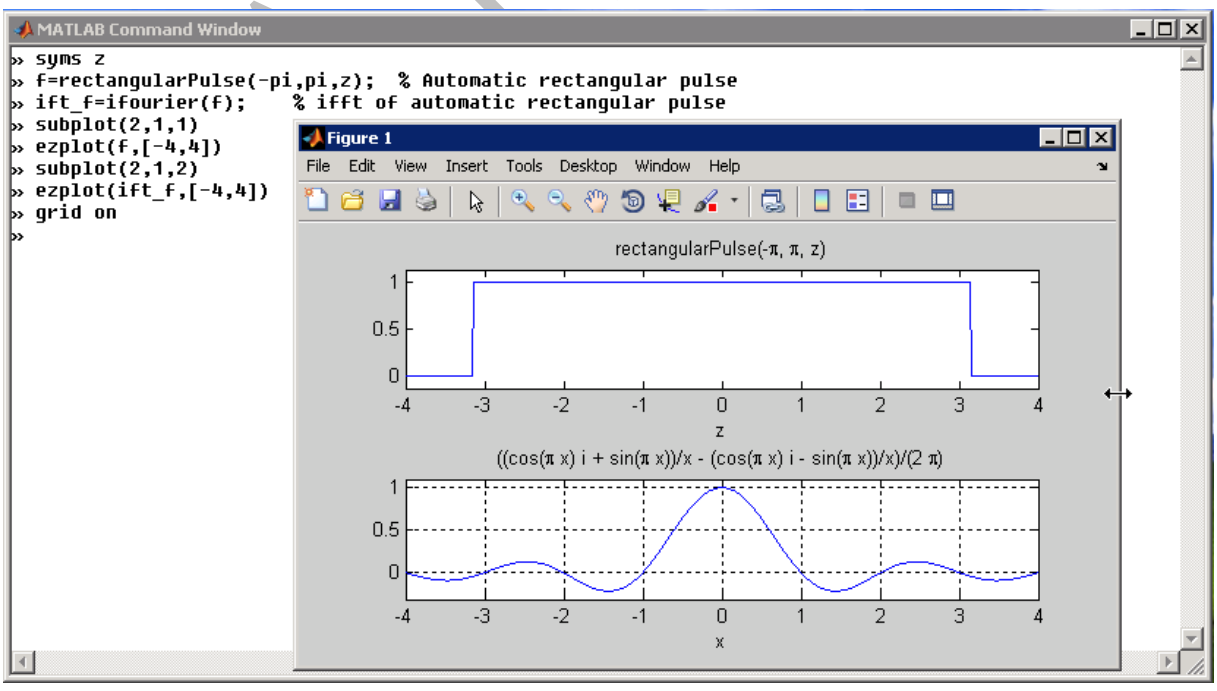
```
MATLAB Command Window
» syms f(t) w
» fourier(diff(f(t), t), t, w)

ans =

w*fourier(f(t), t, w)*i

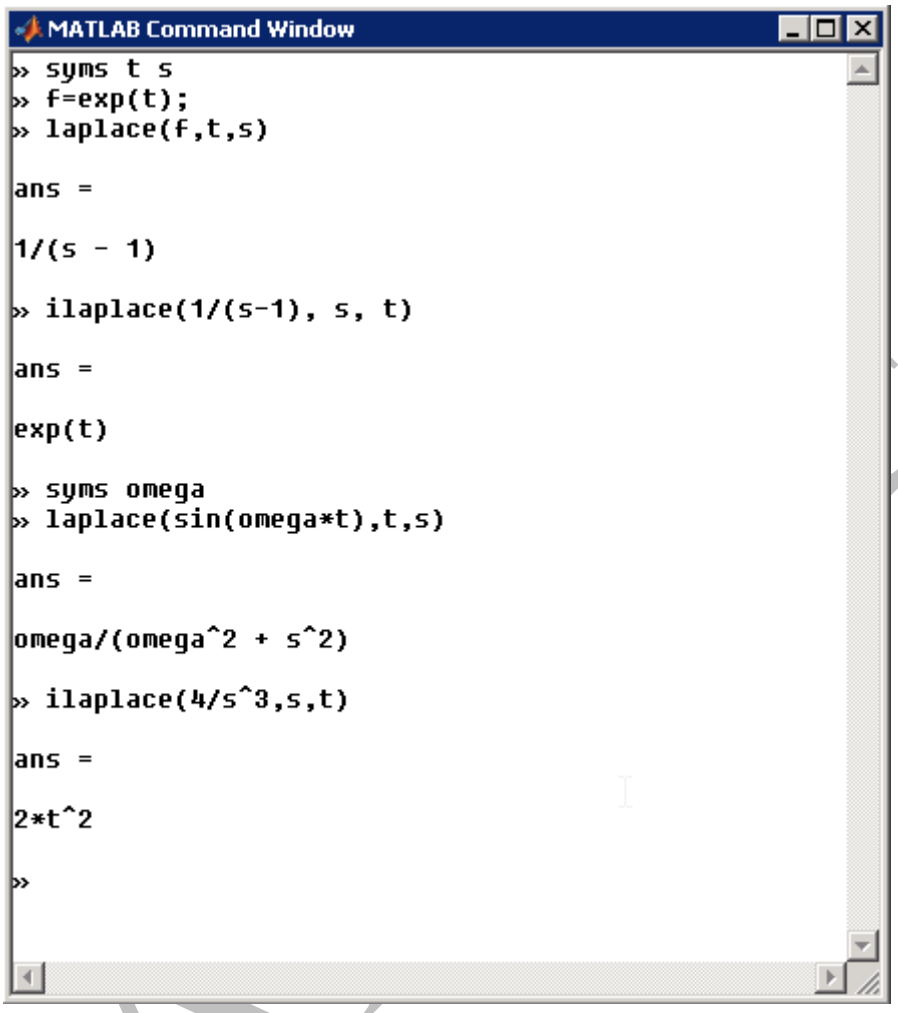
»
```

And another famous typical case that we have calculated by hand in the theoretical course using the function **rectangularPulse( )** also from the Symbolic Math Toolbox.



Laplace Transform

Have a look to some Laplace transformation we calculated by hand in the theoretical course:

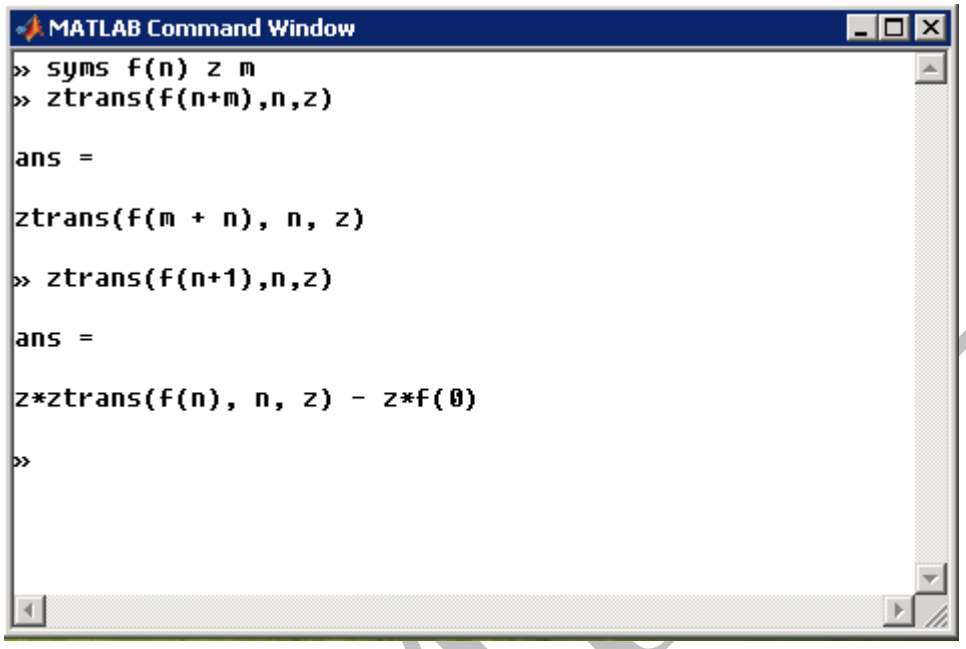


Everything is OK here!

Z Transform

For the Z transform we will focus only on the practical financial case of this transformation that we have study during the theoretical course.

Let's have a look:



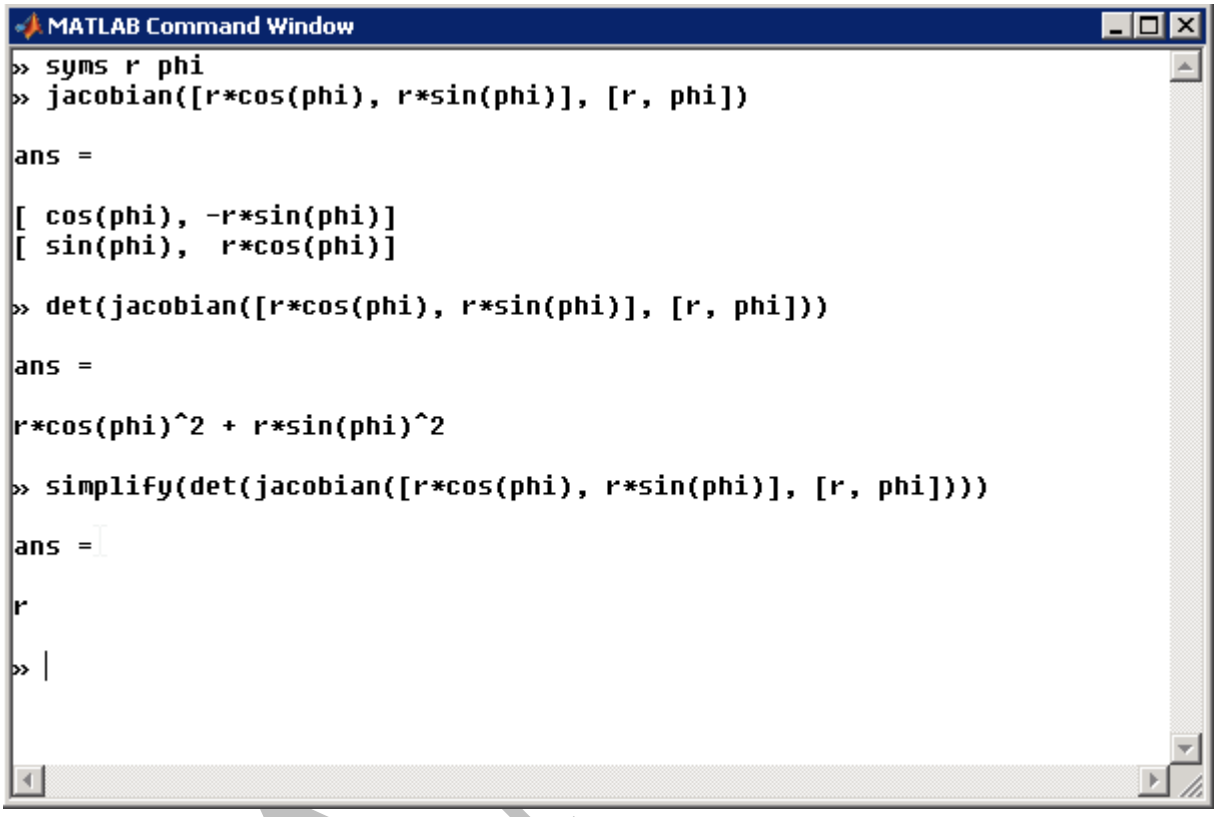
As we can see, MATLAB seems not to be able to manager a general result for any *m* but if we chose a special value than the result is ok.

SAMPLE

Jacobien

As we saw in the course of applied mathematics the jacobian has an important place in some integrals and even sometimes in some statistical tests.

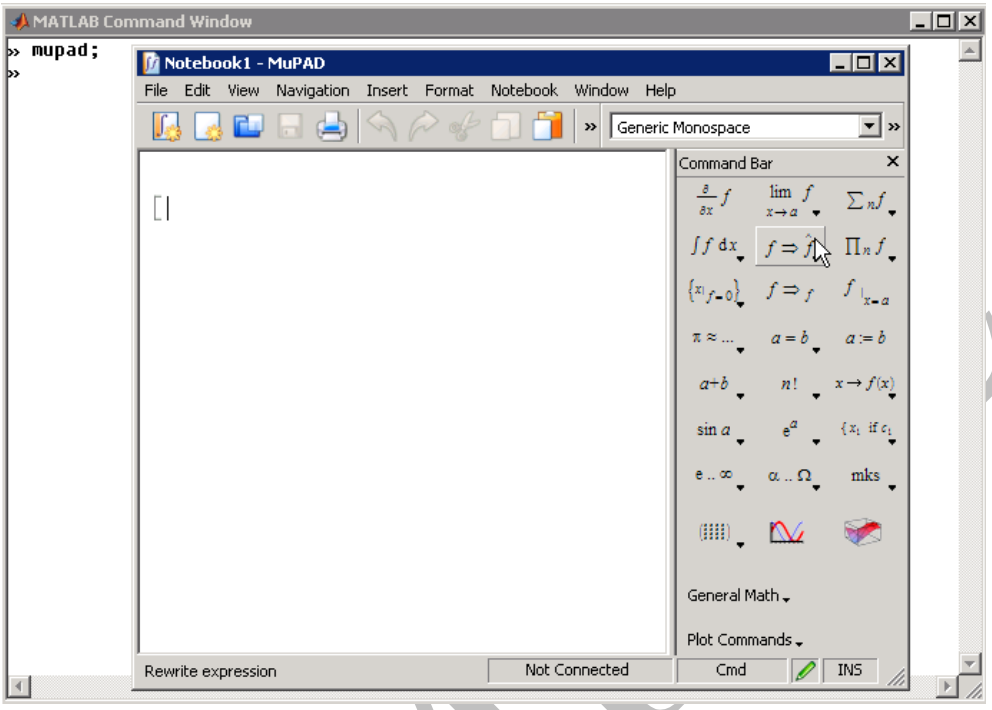
Let us see if we can found back the same result as the one calculate by hand using the function **jacobian( )**:



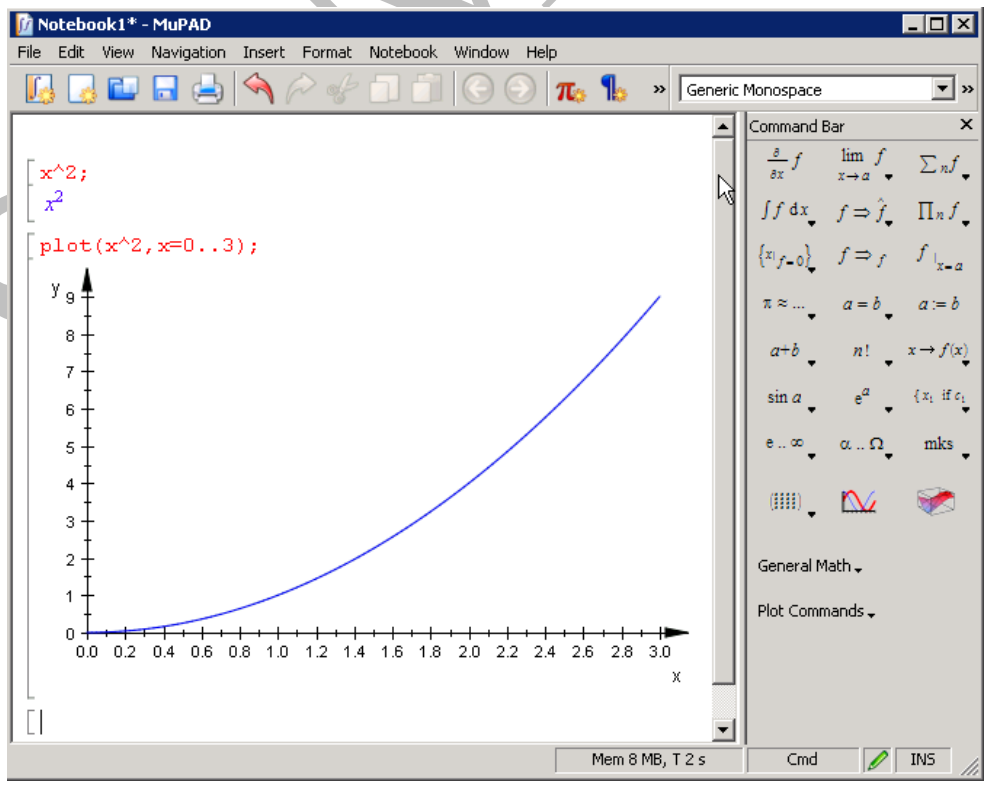
SA

MuPAD

The MuPAD is a Maple like interface in MATLAB to make symbolic calculation and charting. To start it just type **mupad** in the *Command Window*:



And now we just make a small example and as you can see it is very similar to Maple (at least for basics features):



Databases (with Database/Datafeed toolbox)

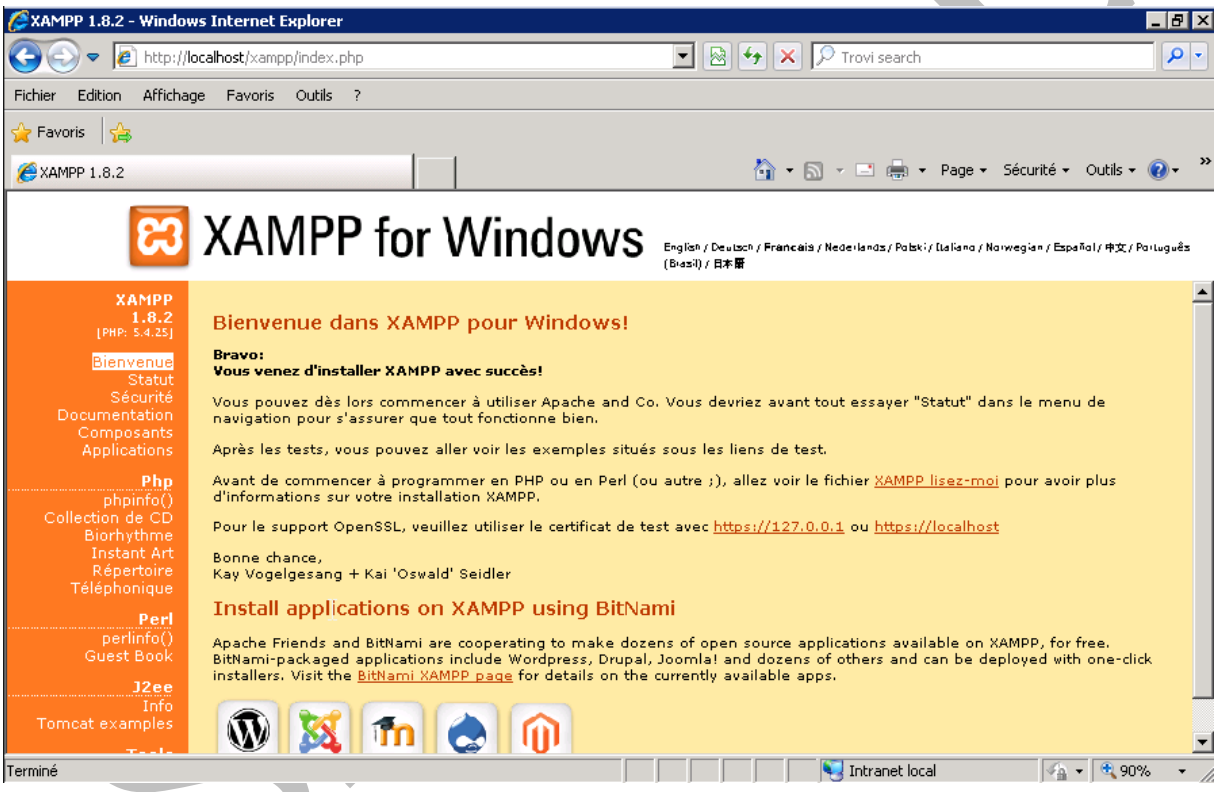
http://www.MathWorks.ch/ch/help/database/release-notes.html

Access to external databases is a major topic in almost all statistical softwares and especially for practitioners in Quantitative Finance, Data Mining and Machine Learning (Big Data).

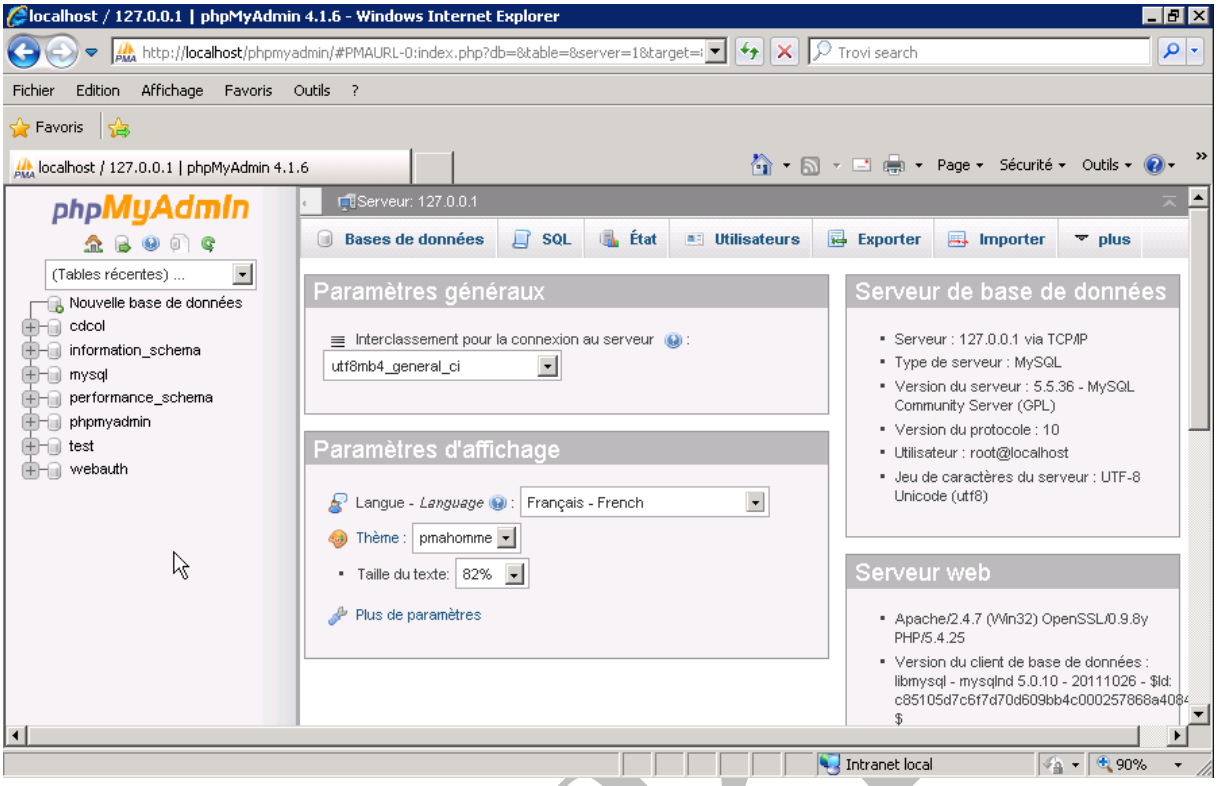
We will see here the most common example to access and retrieve datas form mySQL, Microsoft Access, Oracle Express and SQL Server Express.

Connect to mySQL 5.5.36

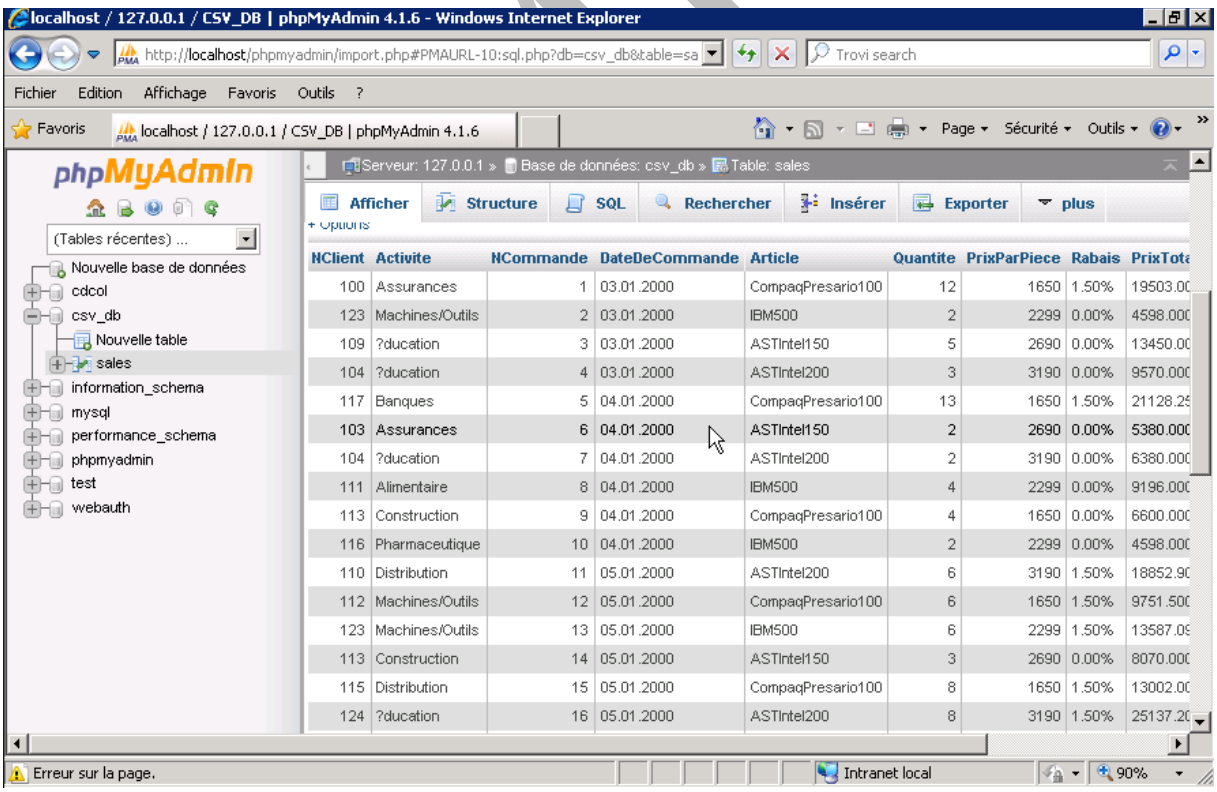
We will begin with mySQL that was for this example installed automatically with XAMPP server:



Then if you click on the mySQL link you get:

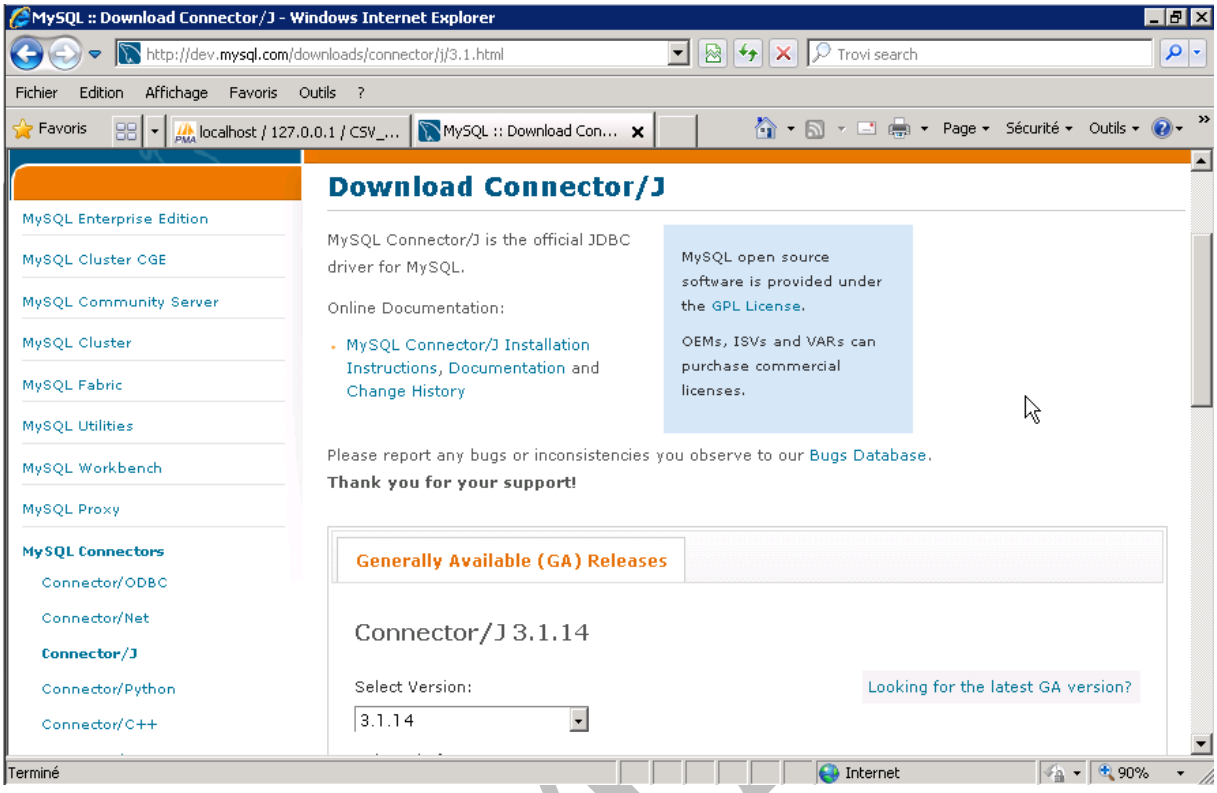


For the example we will import the *sales* table to MATLAB:

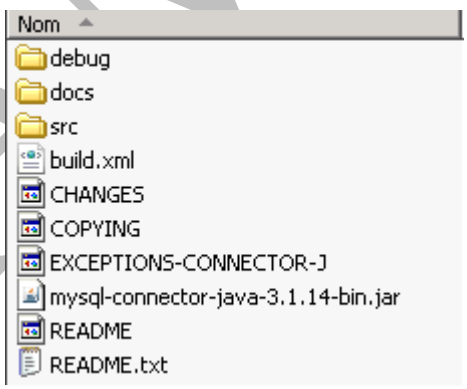


The first thing to do is to install the mySQL JDBC connector of the website:

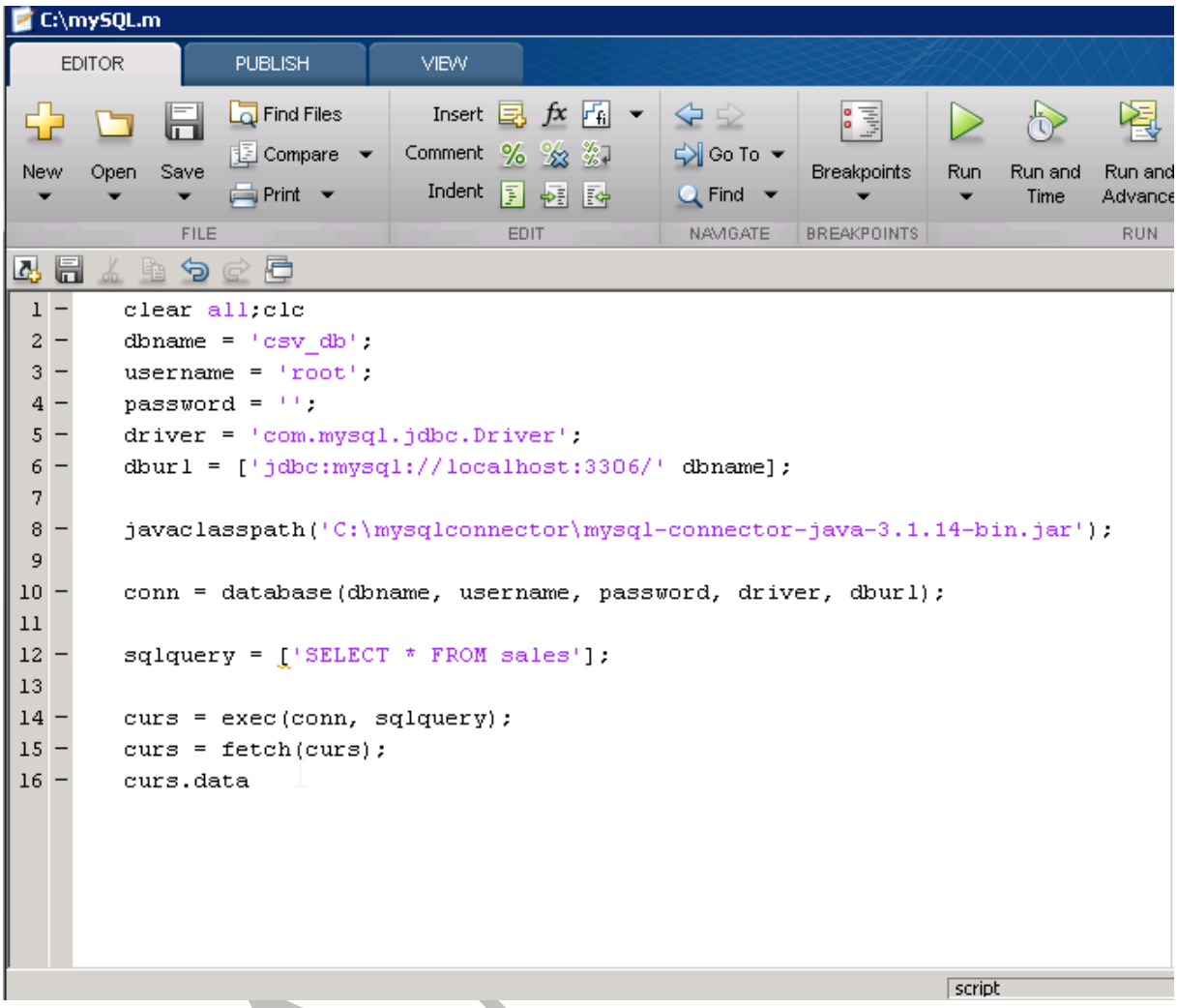
http://dev.mysql.com/download/connector/j/3.1.hmtl



Once downloaded and the zip file extracted you get a folder with the following files and subfolders:

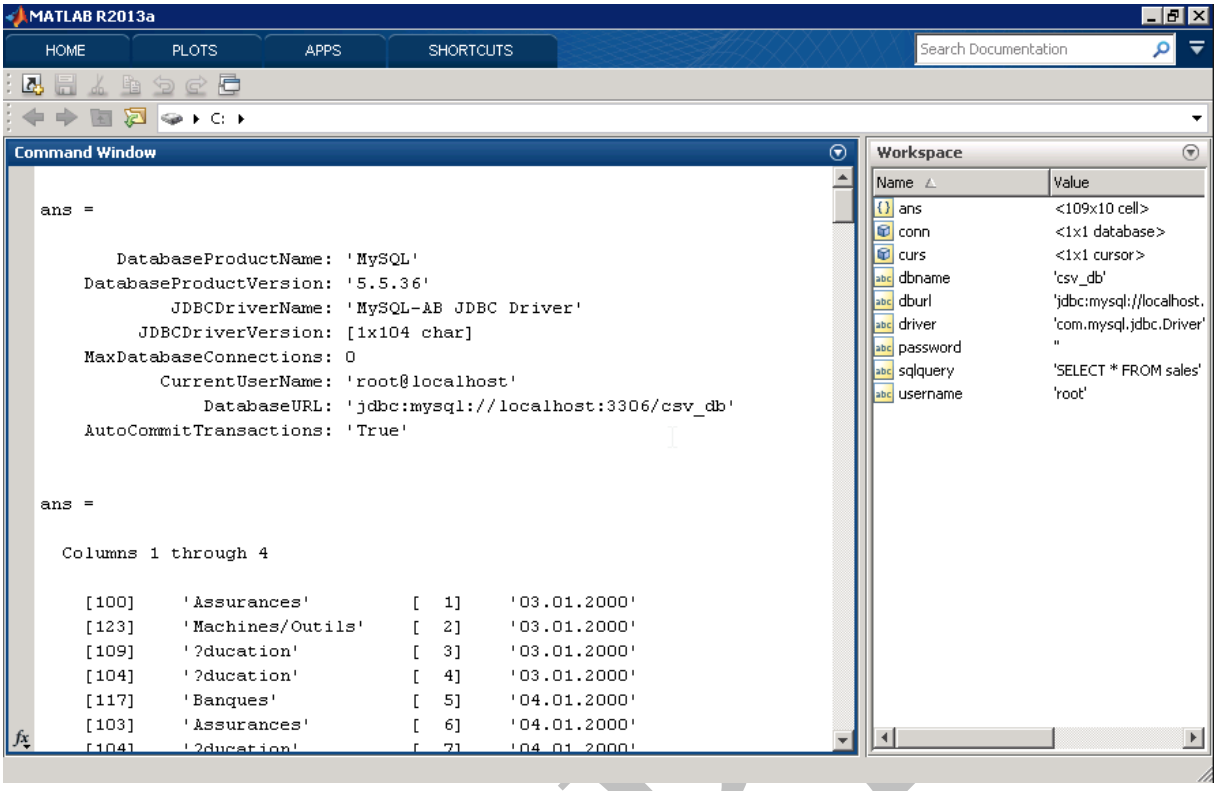


Once the connector unzipped we write the following script:



Once the code is executed the datas are imported in a cell array (**ToDo**: found a way to directly import the data into a dataset).

SAMPLE

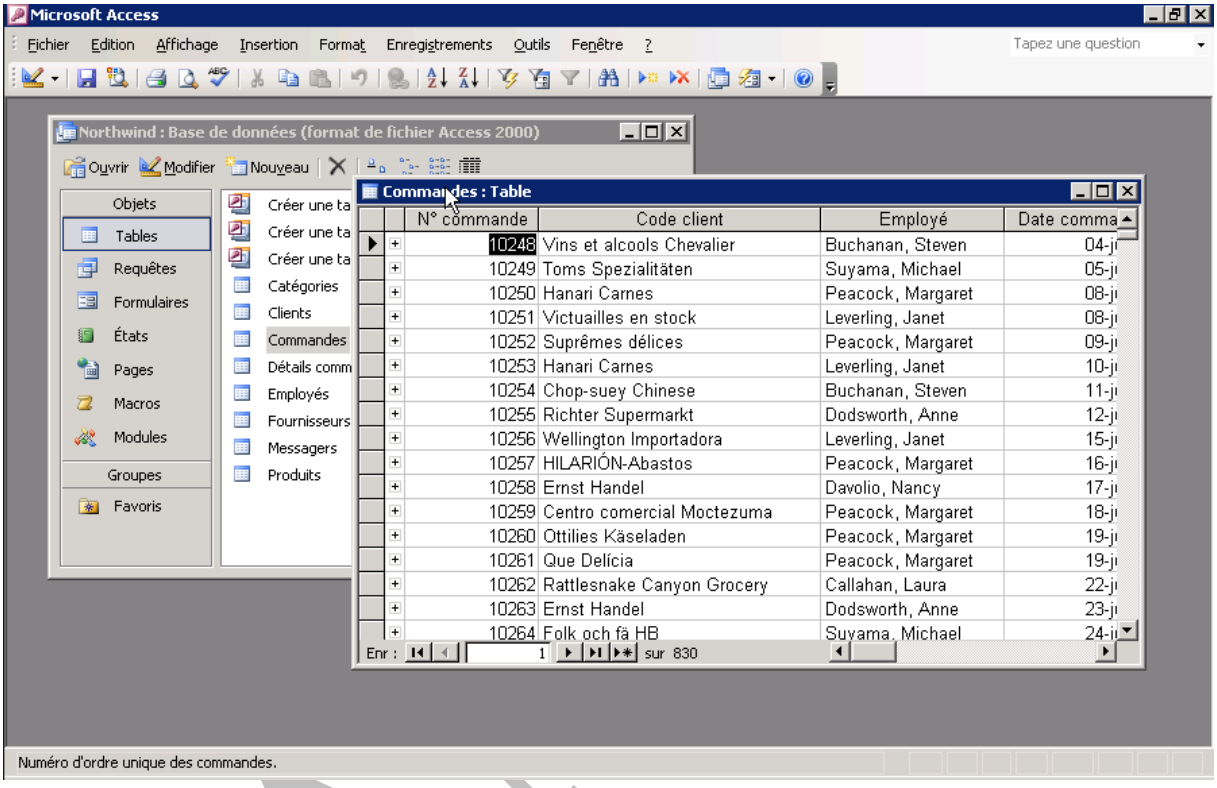

MATLAB R2013a
HOME
PLOTS
APPS
SHORTCUTS
Search Documentation
C:
Command Window
ans =
DatabaseProductName: 'MySQL'
DatabaseProductVersion: '5.5.36'
JDBCDriverName: 'MySQL-AB JDBC Driver'
JDBCDriverVersion: [1x104 char]
MaxDatabaseConnections: 0
CurrentUserName: 'root@localhost'
DatabaseURL: 'jdbc:mysql://localhost:3306/csv_db'
AutoCommitTransactions: 'True'
ans =
Columns 1 through 4
[100] 'Assurances' [1] '03.01.2000'
[123] 'Machines/Outils' [2] '03.01.2000'
[109] '?ducation' [3] '03.01.2000'
[104] '?ducation' [4] '03.01.2000'
[117] 'Banques' [5] '04.01.2000'
[103] 'Assurances' [6] '04.01.2000'
Workspace
Name
Value
ans <109x10 cell>
conn <1x1 database>
curs <1x1 cursor>
dbname 'csv_db'
dburl 'jdbc:mysql://localhost.
driver 'com.mysql.jdbc.Driver'
password ''
sqlquery 'SELECT * FROM sales'
username 'root'

SAMPLE

Connect to Microsoft Access 2003/2010 (*.mdb/*.accdb)

Retrieve data from Microsoft Access is something very common even Fortune 500 companies use Microsoft Access with MATLAB...

To start the first example let us see how to connect a *.mdb (Microsoft DataBase file) with MATLAB. Consider for this purpose the following Northwind.mdb file and especially the table named *Commandes*:

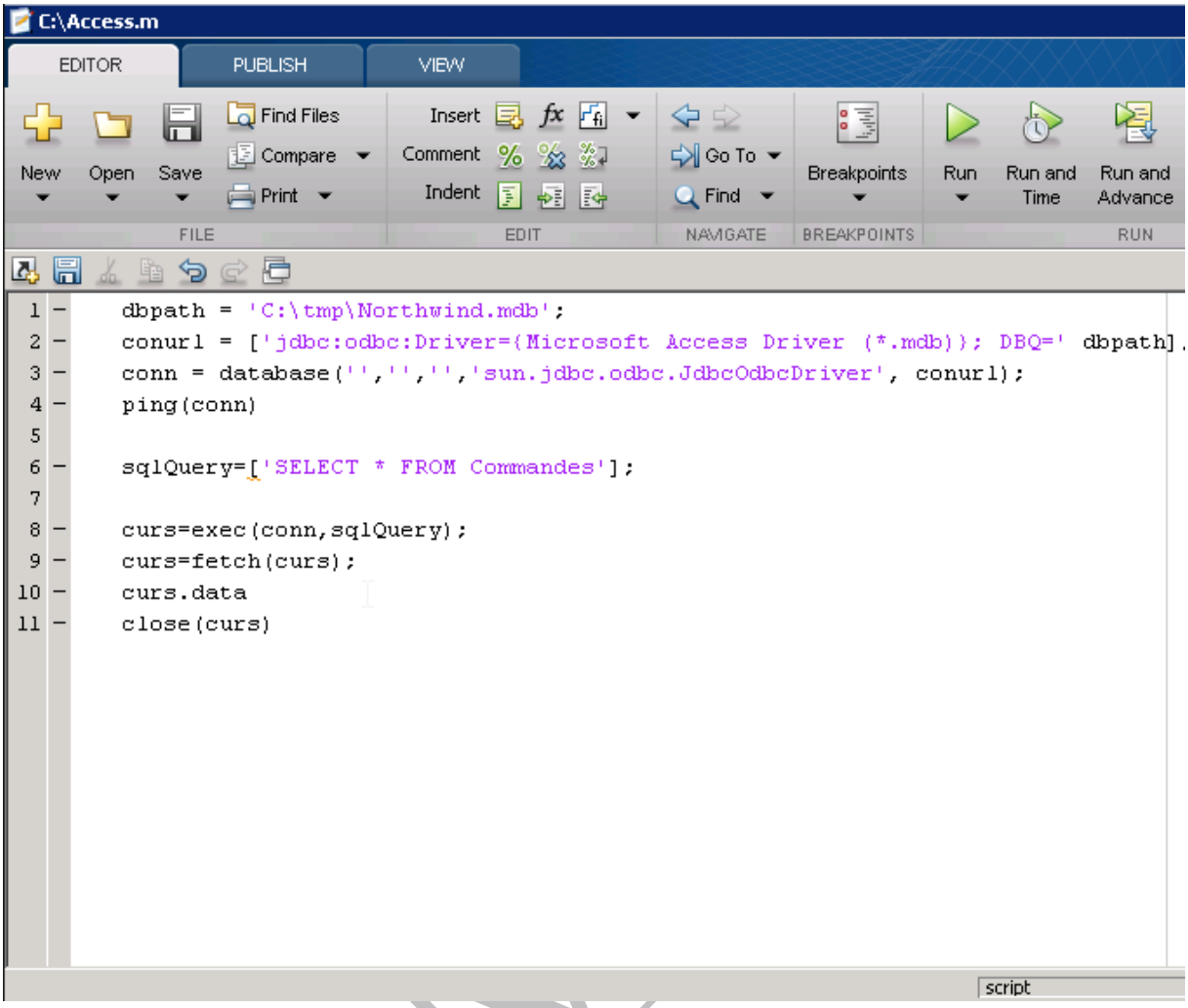


To retrieve data from such a table you need the following script but you must first be sure that you have at least the Java Runtime installed:

https://www.java.com/fr/download/

and the Microsoft Access ODBC Driver:

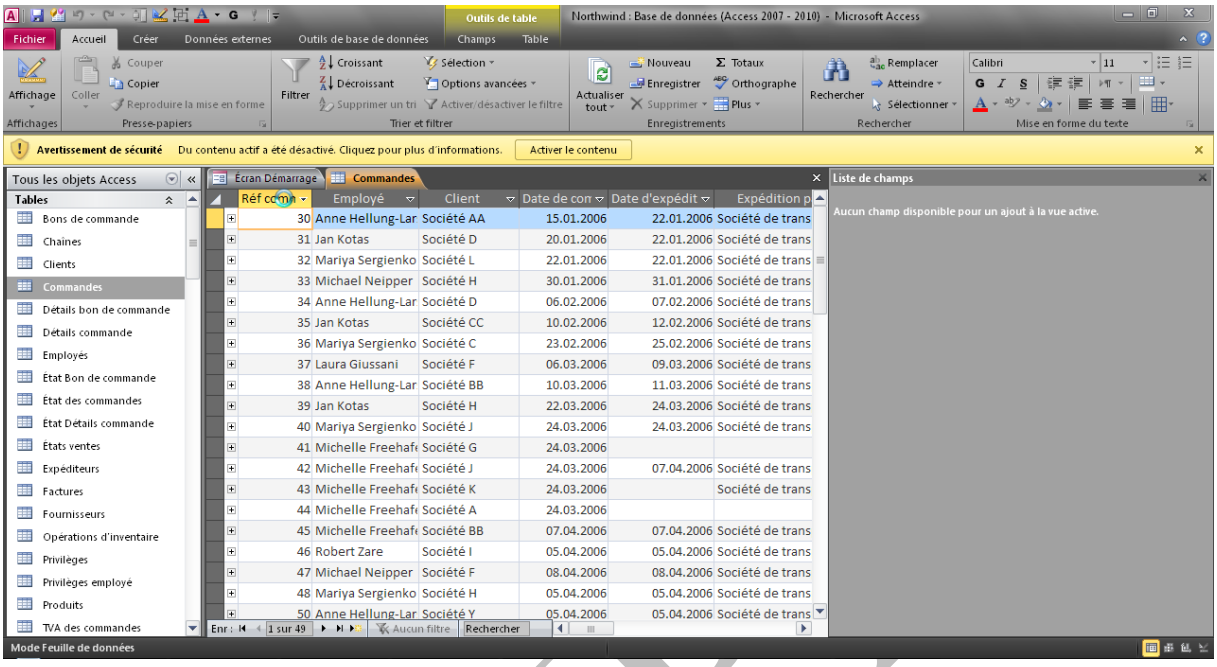
http://www.microsoft.com/en-us/download/details.aspx?id=13255



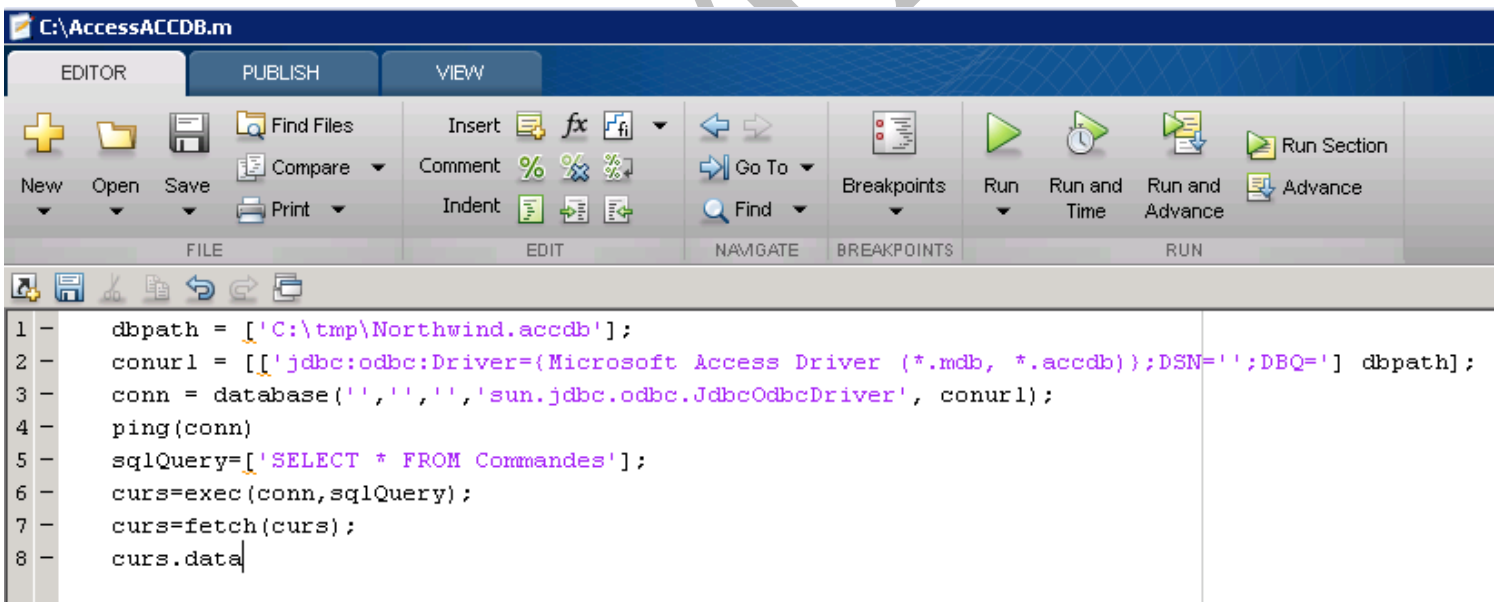
The imported result will also be a cell array (**ToDo**: found a way to directly import the data into a dataset).

SAMPLE

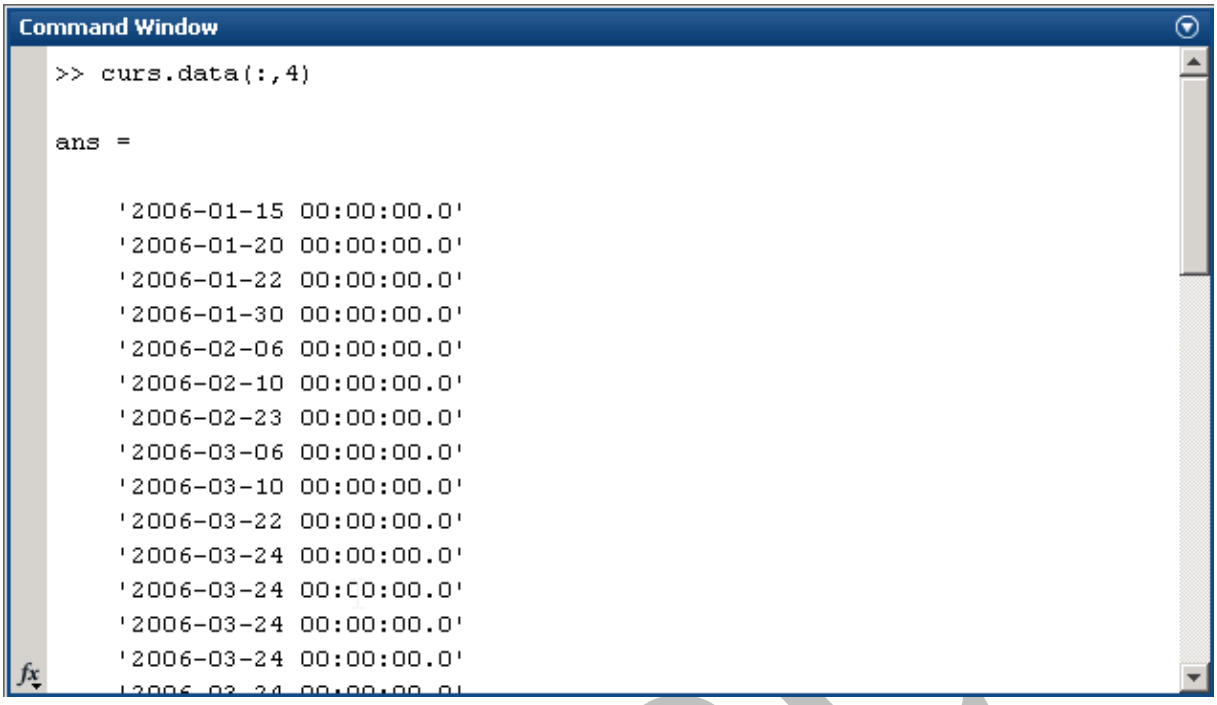
For .*accdb files (it can help to have ODBC driver installed and an up to date for Microsoft Access 2007/2010/2013 version and sometimes even having the MS Access runtime installed!) with the same table:



The following script will make the job (you must also have the Java Runtime and the Microsoft Access ODBC Driver installed!):



and the result will always be (**ToDo**: found a way to directly import the data into a dataset):

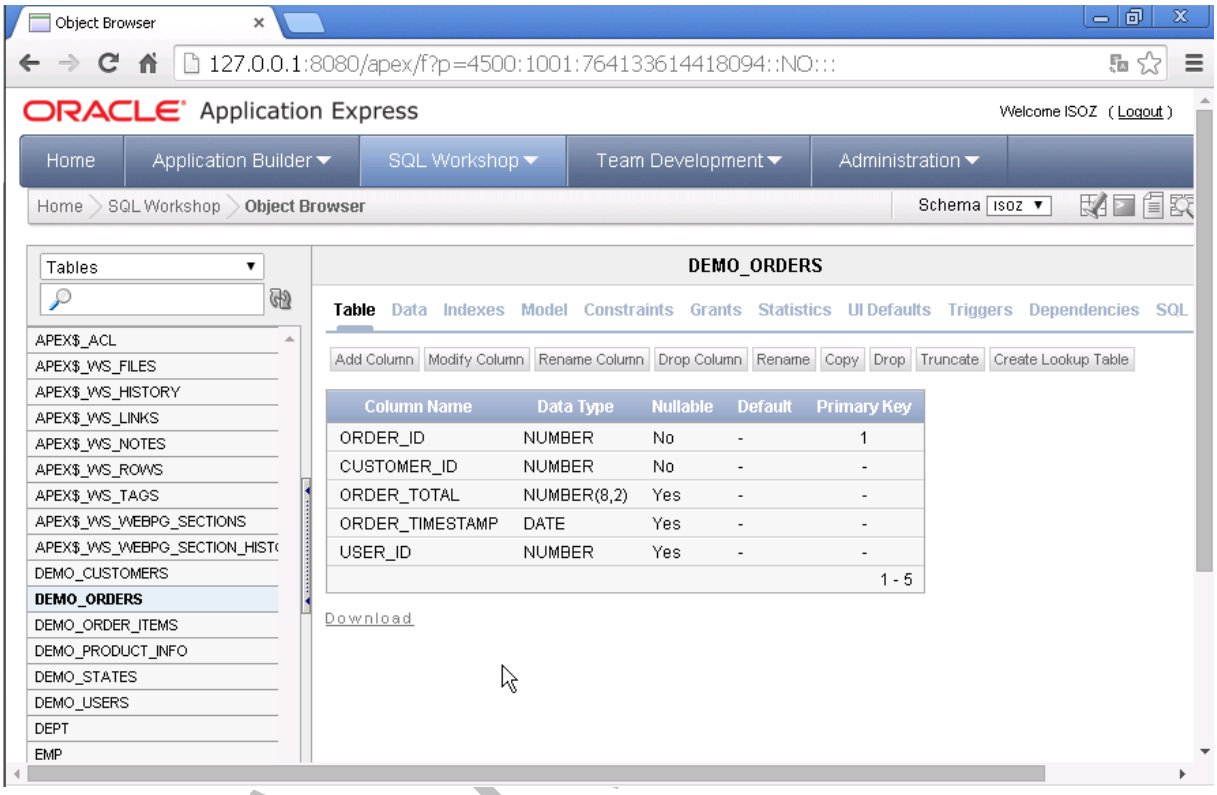

Command Window
>> curs.data(:,4)
ans =
'2006-01-15 00:00:00.0'
'2006-01-20 00:00:00.0'
'2006-01-22 00:00:00.0'
'2006-01-30 00:00:00.0'
'2006-02-06 00:00:00.0'
'2006-02-10 00:00:00.0'
'2006-02-23 00:00:00.0'
'2006-03-06 00:00:00.0'
'2006-03-10 00:00:00.0'
'2006-03-22 00:00:00.0'
'2006-03-24 00:00:00.0'
'2006-03-24 00:C0:00.0'
'2006-03-24 00:00:00.0'
'2006-03-24 00:00:00.0'

SAMPLE

Connect to Oracle Express 11g

Oracle is in my point of view the most powerful database worldwide (and also my preferred DBMS).

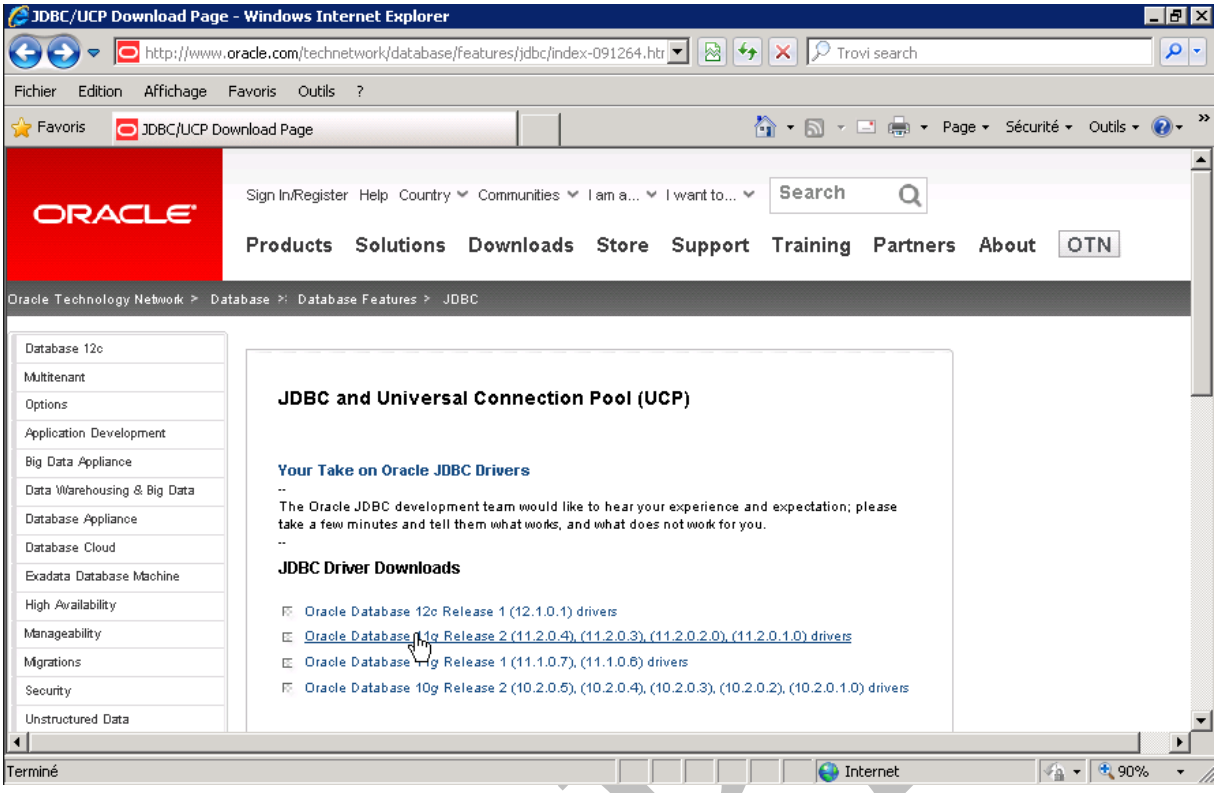
To see how to retrieve data from Oracle (Express) 11g we will use the native *DEMO_ORDERS* table:



First we will need to download the Oracle JDBC driver on the Oracle website:

http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html

SAMPLE

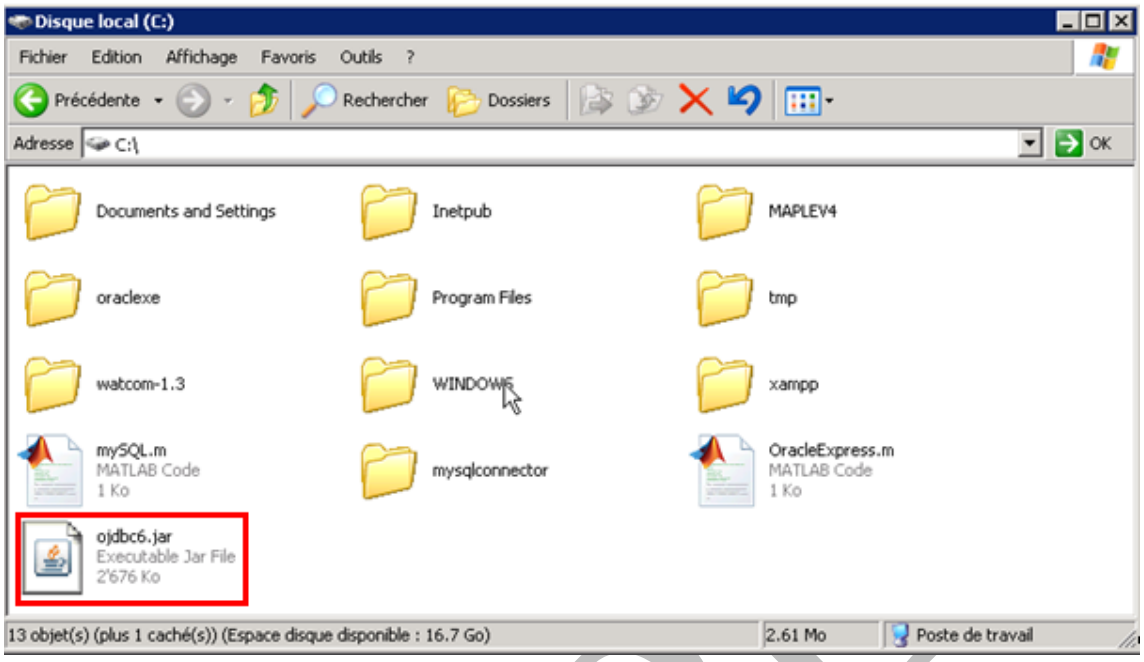


You click on the link **Oracle Database 11g Release 2 (11.2.0) drivers** as you can see above. After you get:

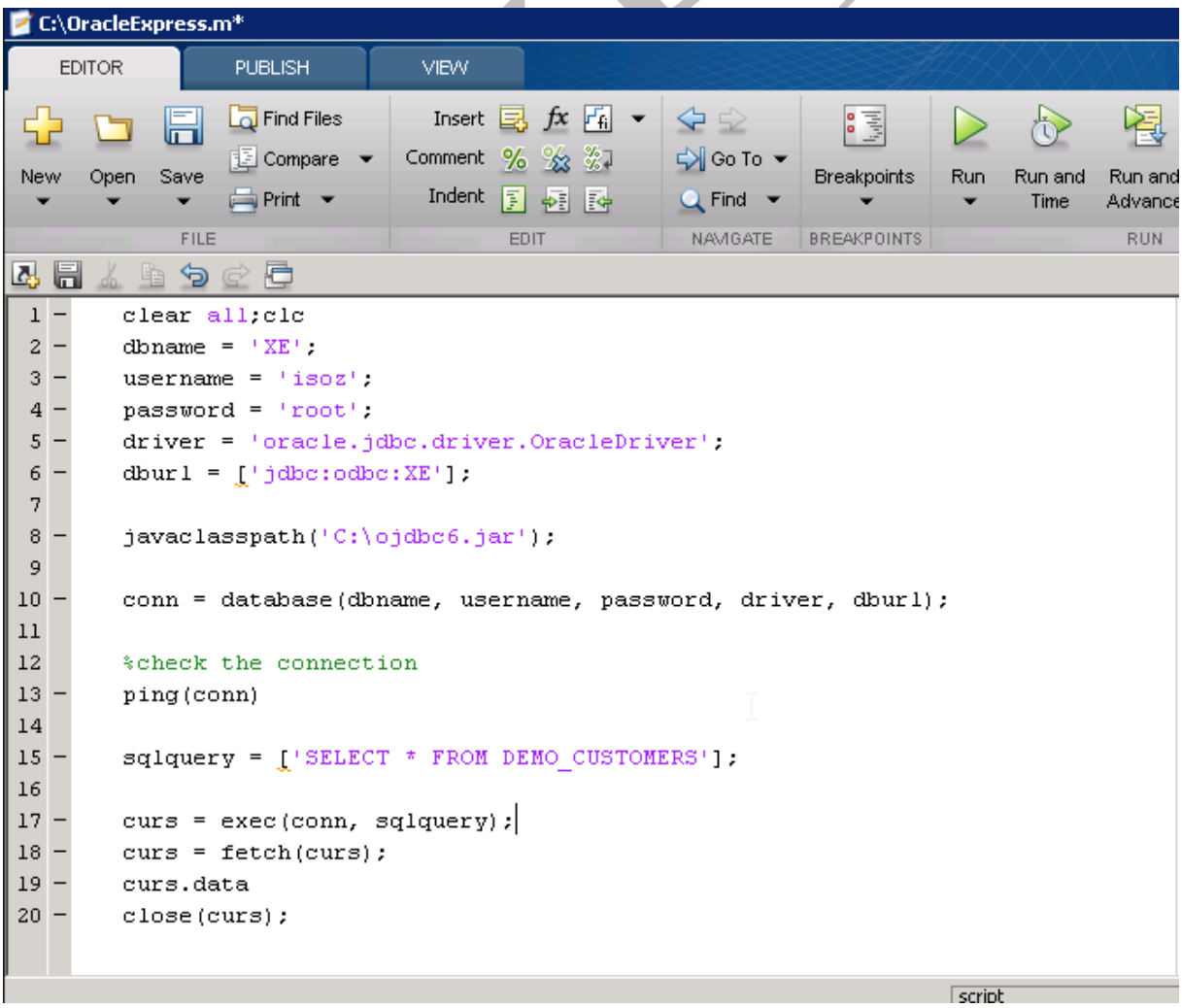


and you click on the ojdbc6.jar file to download.

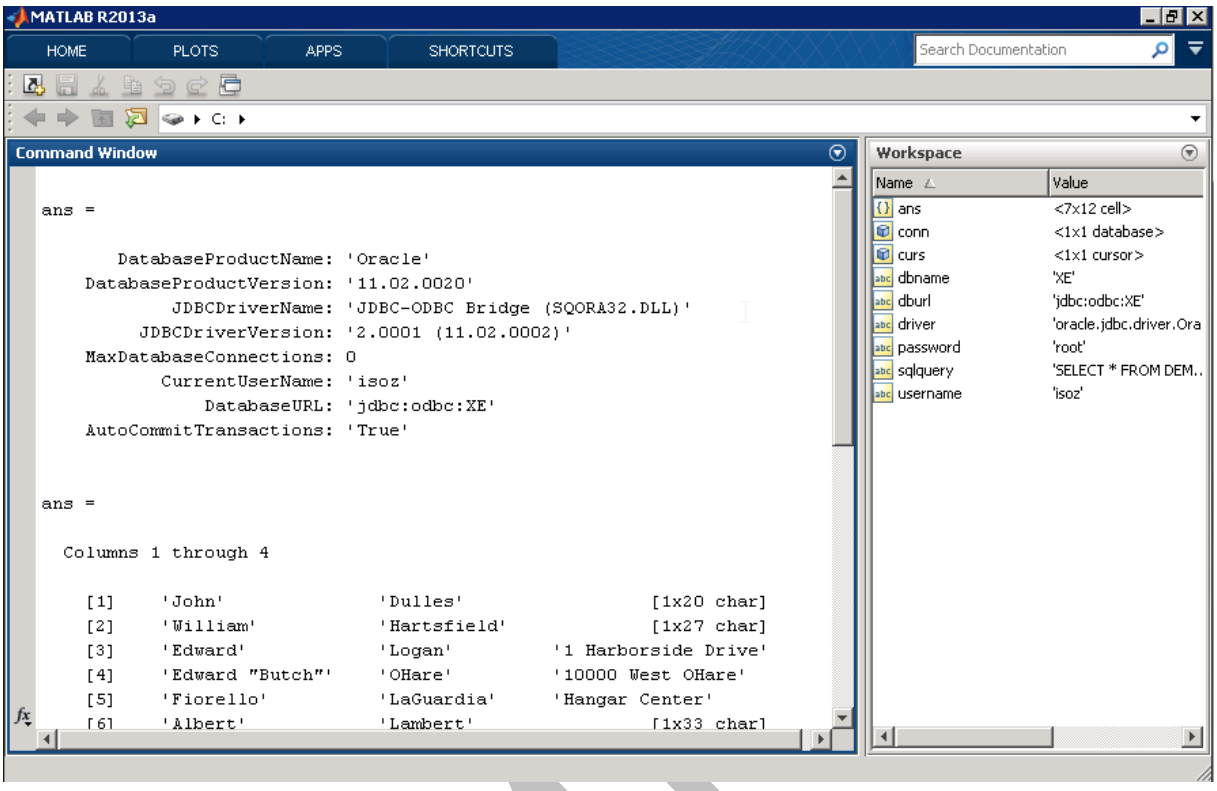
Once downloaded just put the file where you want:



and write the following script (**ToDo**: found a way to directly import the data into a dataset):



The result will also be cell array structure:



SAMPLE

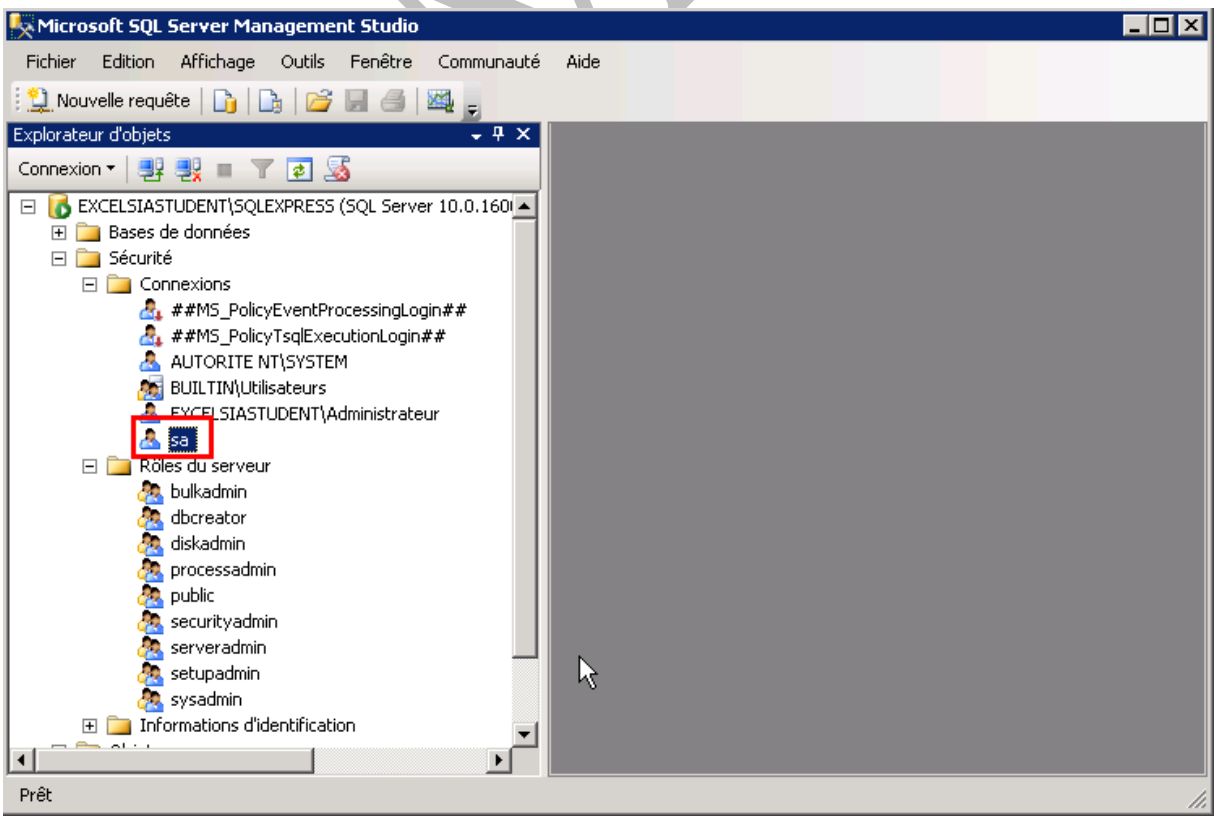
Connect to MS SQL Server Express 2008

Even if it is far away to be the best DBMS, Microsoft SQL Server (Express or not) is largely deployed across the world and even in financial institutions.

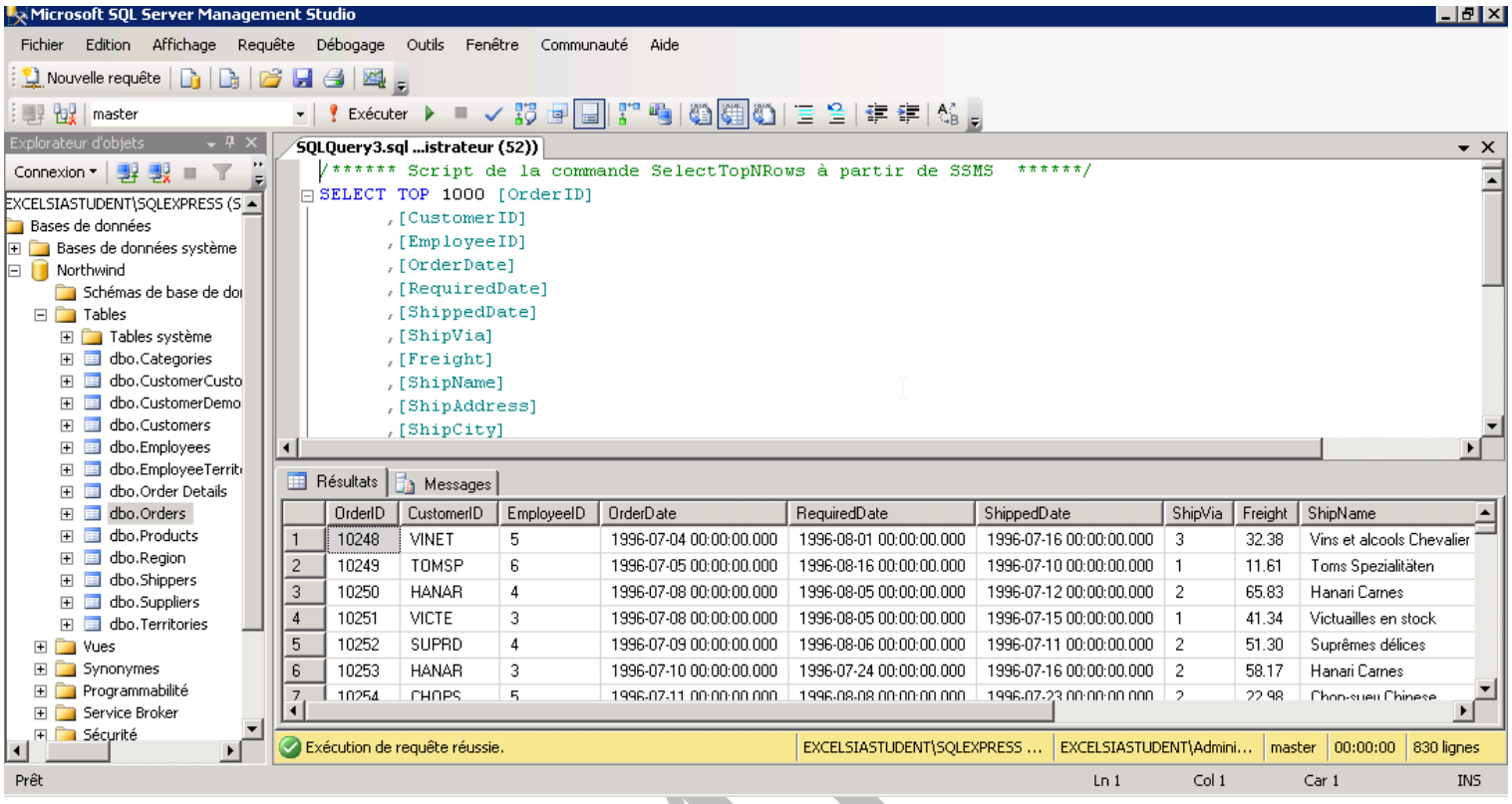
For this example we will use the default SQLEXPRES server name instance:



Below you can the existing system administrator account inside of **SQL Server Express Management Studio** that we will have to use later:

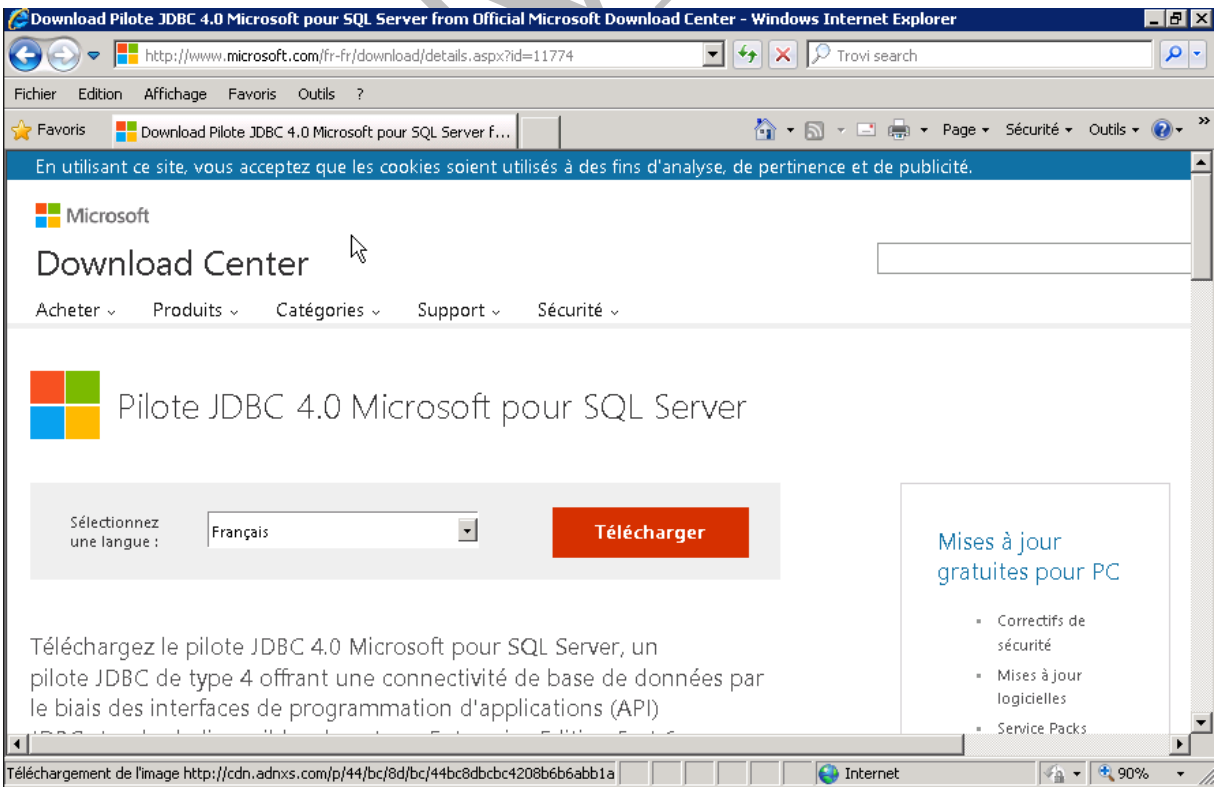


For the example we will use the *Northwind* database and especially the table *dbo.Orders*:

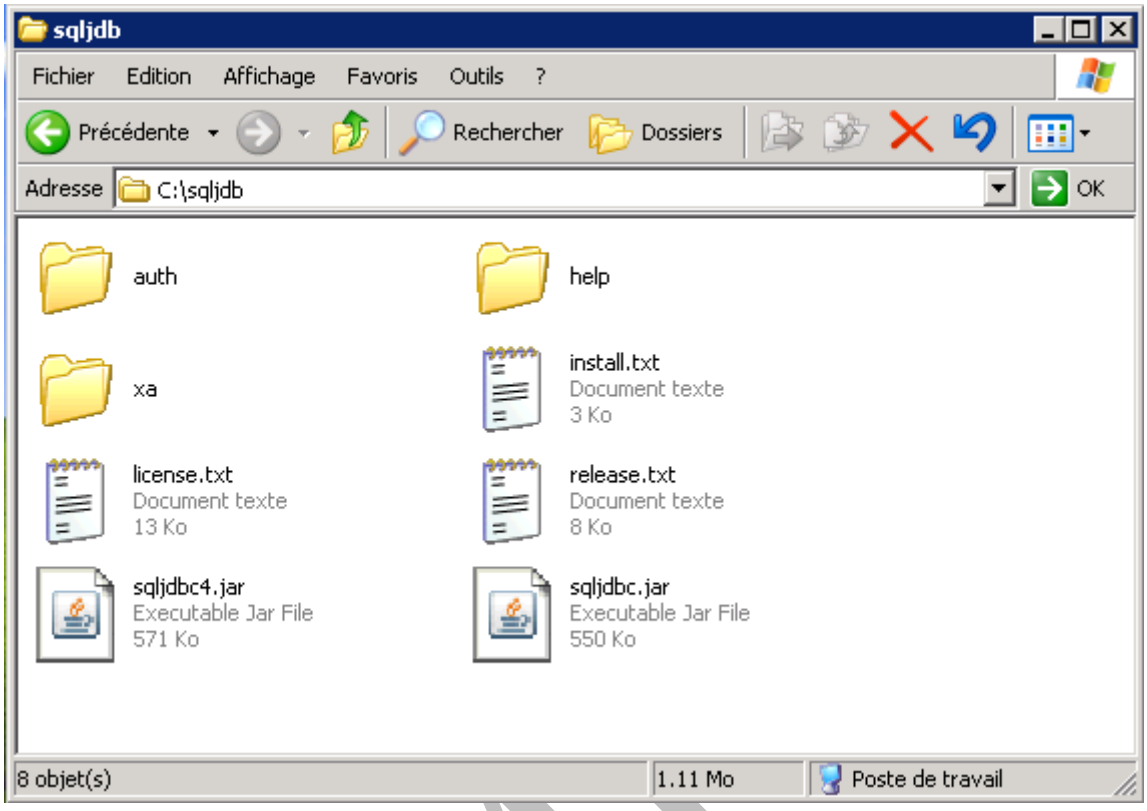


After we need to download the JDBC 4.0 Microsoft SQL Server Driver:

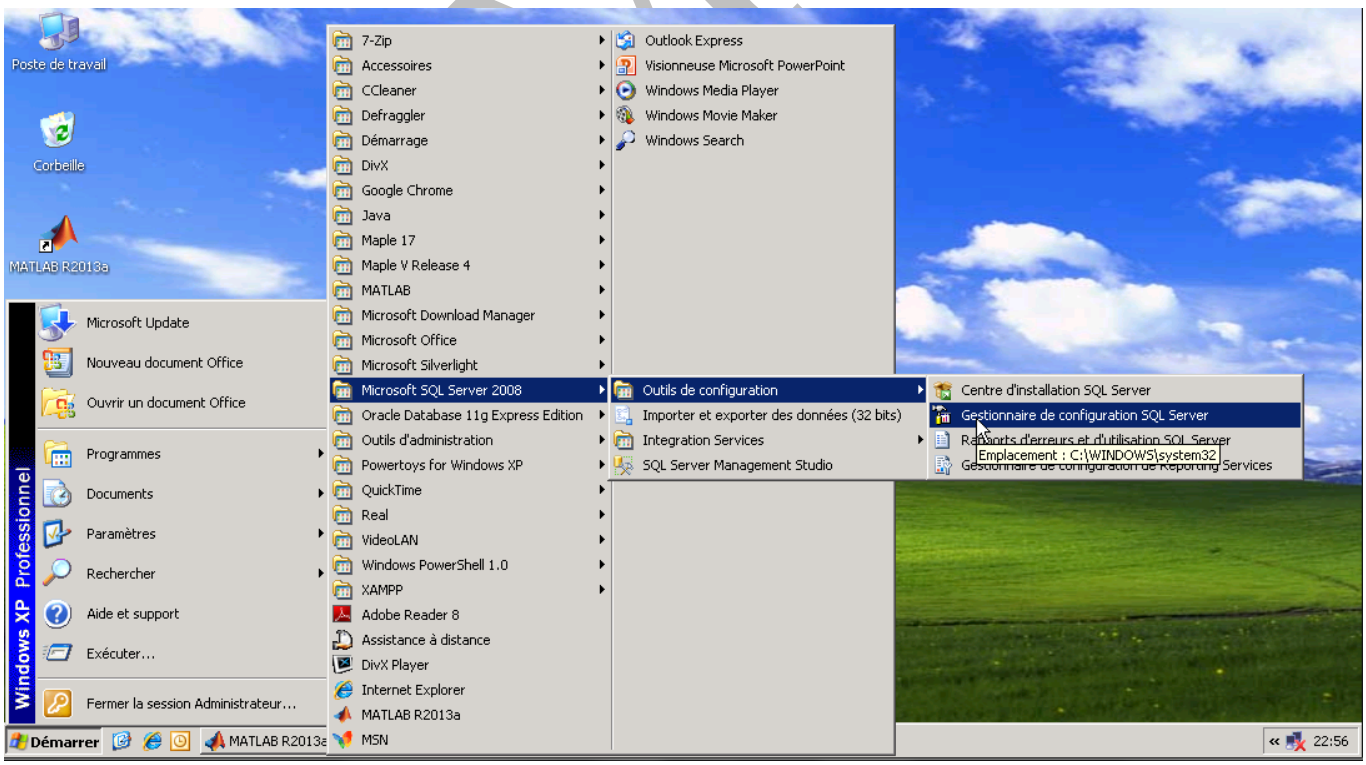
http://www.microsoft.com/fr-fr/download/details.aspx?id=11774



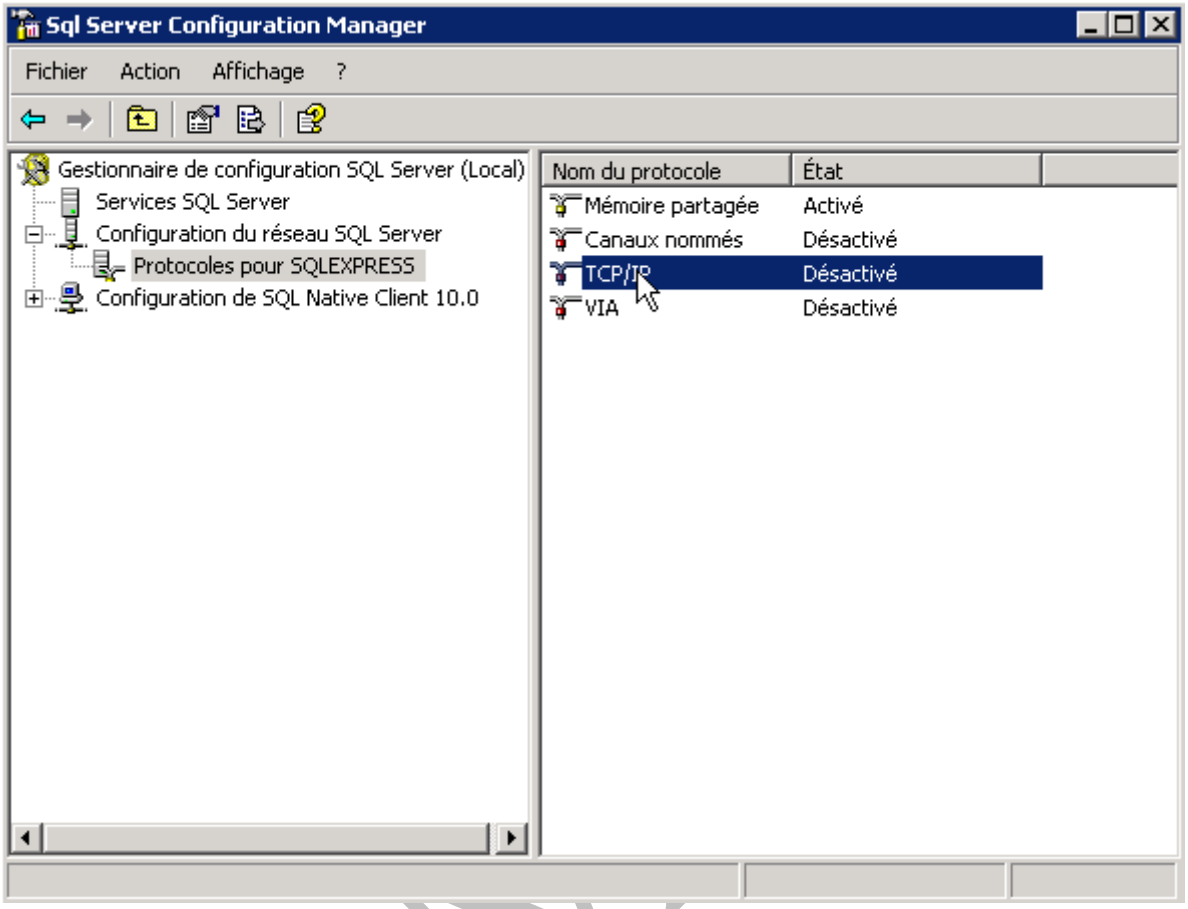
Once de file download and extract you will get the following folder with a few various files:



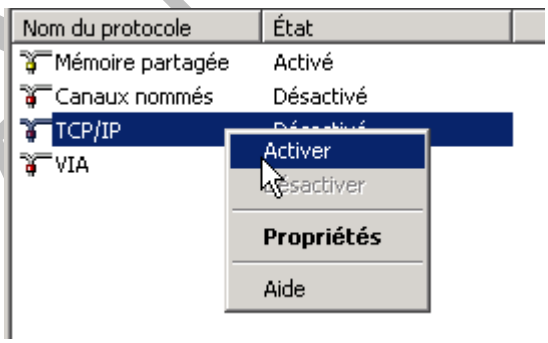
Now we open the **SQL Server Configuration Manager**:



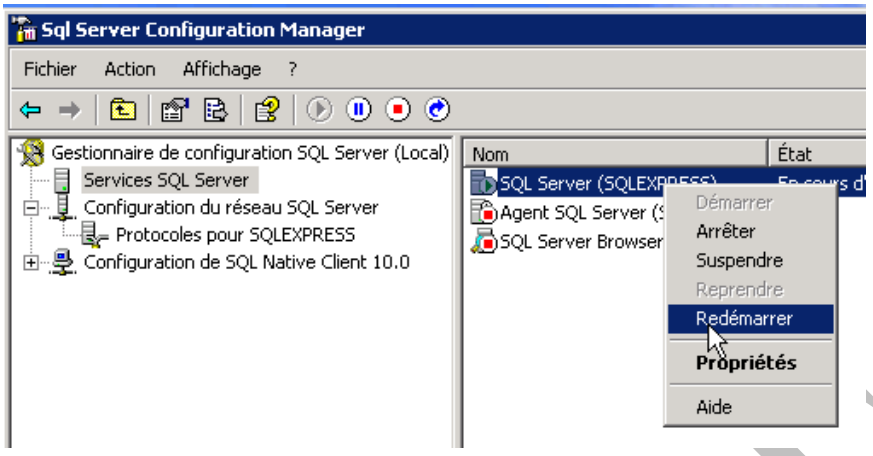
In the windows that will open we select the SQL **Protocoles** branch and on the right the **TCP/IP** protocol:



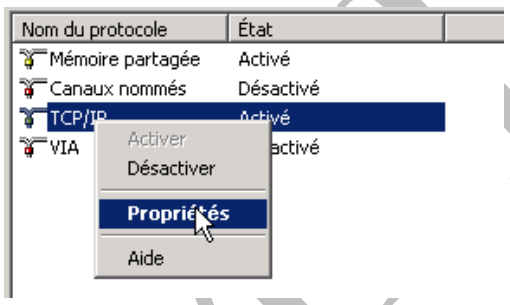
First we need to activate it by making a right click and selecting **Activate**:



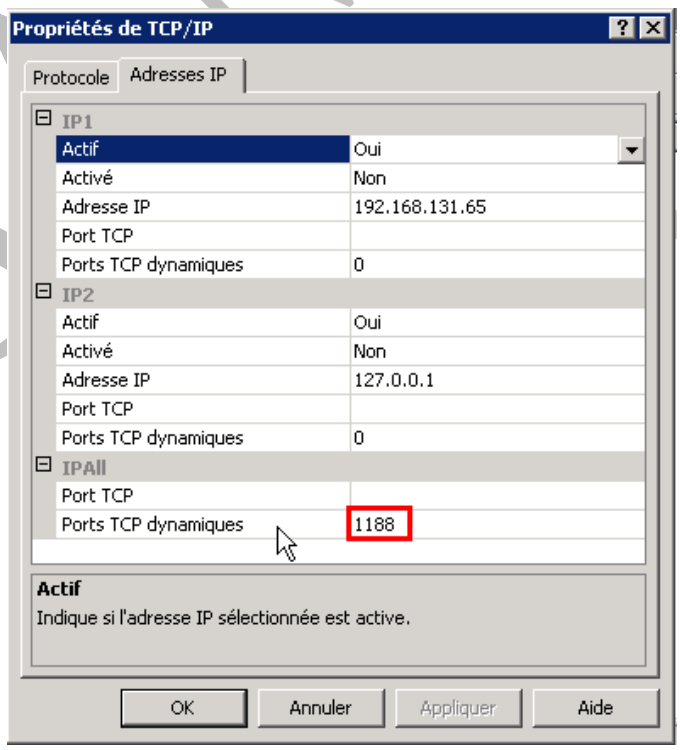
One the TCP/IP activated you must restart the SQL Server in the **Services** branch by right clicking a selecting **Restart**:



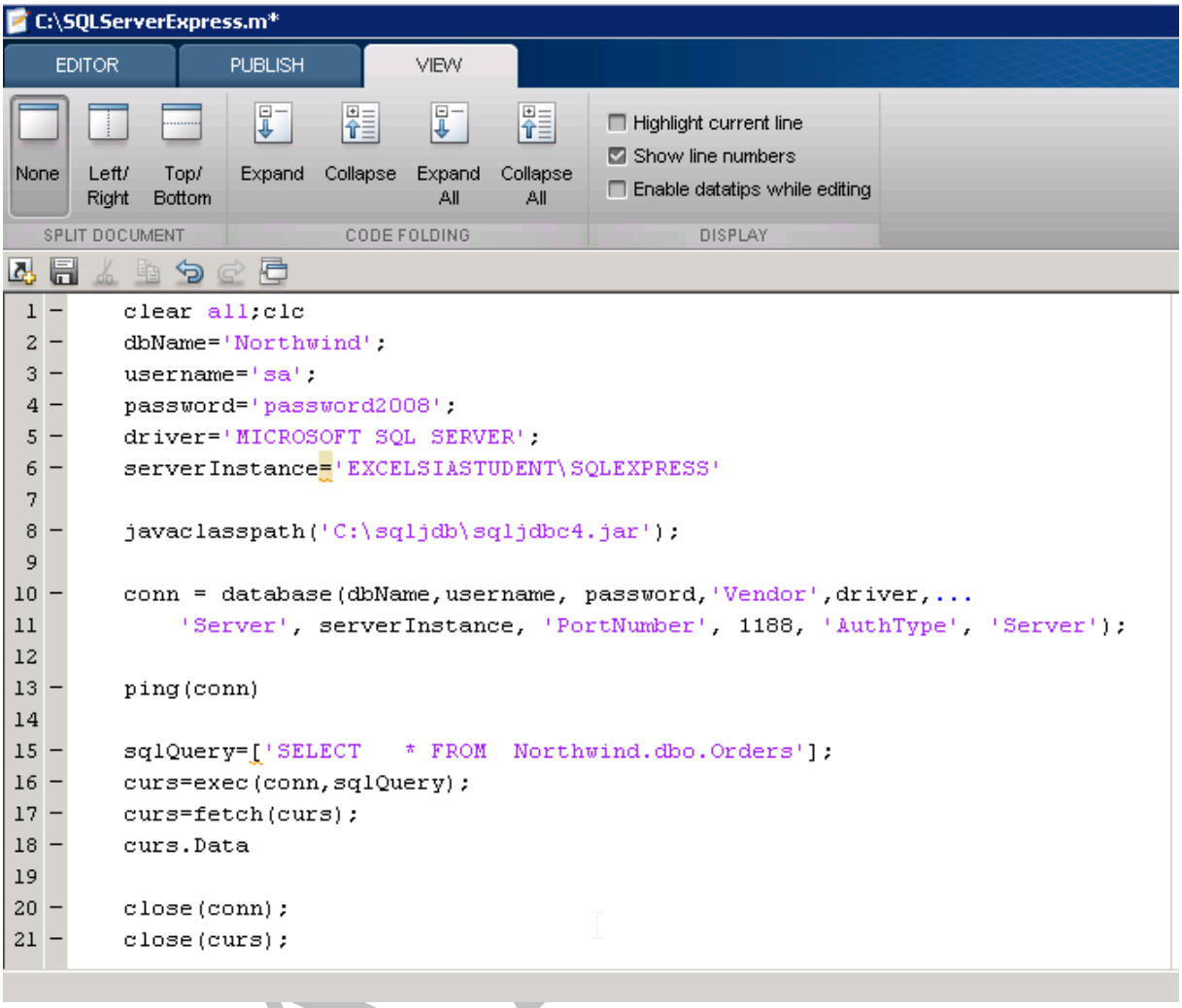
Once this is done we go back to the TCP/IP protocol and we open the **Properties**:



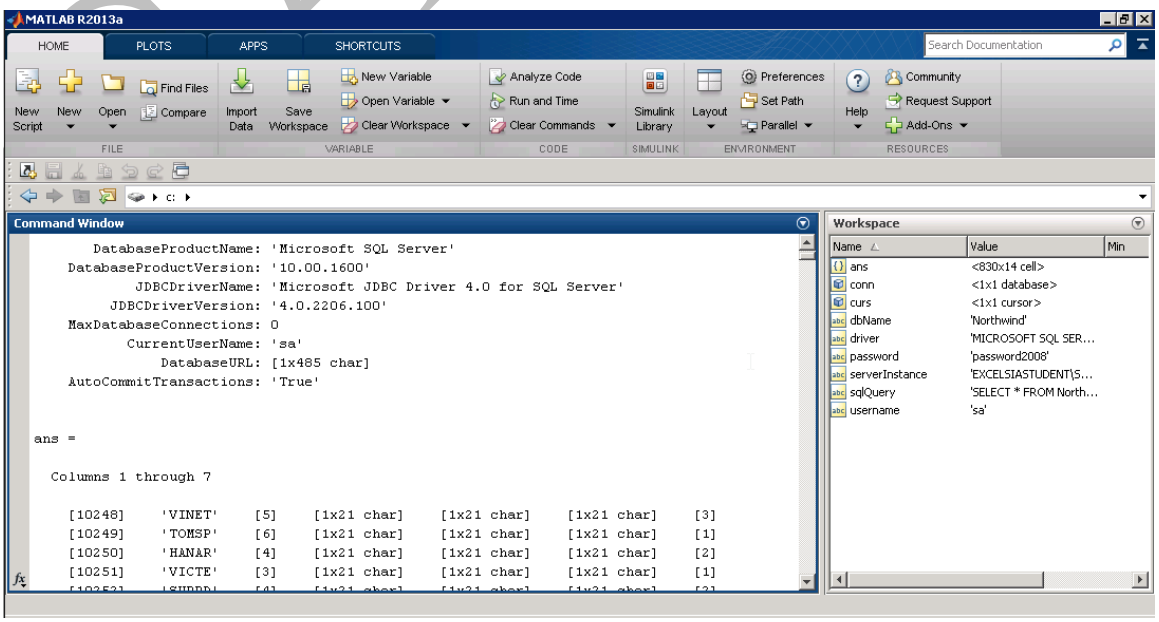
The following windows will open and it is important to take note of the **TCP dynamic port** or to write a given one if the field is empty:



Now we write the following script to retrieve the data (**ToDo**: found a way to directly import the data into a dataset):



And the result will also be a cell array structure:



Connect to Yahoo! Finance

Here we use the **fetch( )** function of the database 2013a toolbox to retrieve data directly from Yahoo! Finance into MATLAB and plot them:

```
close all; clear all; clc;

date_from=datenum('Jan 1 2009');
date_to=datenum('Aug 27 2013');

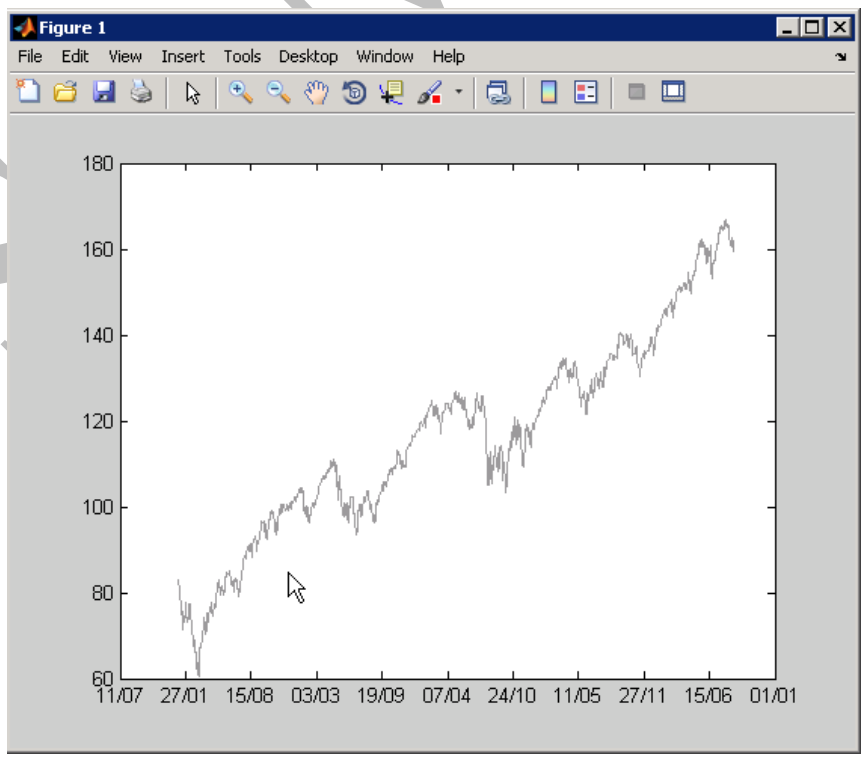
stock='SPY';

%we retrieve the close values (with the dates in the first column)
adjClose = fetch(yahoo,stock,'adj close',date_from,date_to);
%we retrieve the divident values (with the dates in the first column)
div = fetch(yahoo,stock,date_from,date_to,'v');

%we calculate the returns
returns=(adjClose(2:end,2)./adjClose(1:end-1,2)-1);

% We plot adjusted close price of and mark days when dividends
% have been announced
plot(adjClose(:,1),adjClose(:,2),'color',[0.6 0.6 0.6])
datetick('x','dd/mm','keepticks');
```

The result gives (we already saw how to format dates in the chart chapter):



Statistics (with Statistical and Curve Fitting Toolbox)

http://www.MathWorks.ch/ch/help/stats/release-notes.html

We will focus now on the Statistics Toolbox of MATLAB but only on the mathematical tools and theorems that we have studied and proved in details during the theoretical course on statistics/data mining and machine learning.

Datasets

The first thing to know with the Statistical Toolbox is that you have the possibility to import or to convert external data to a structure named "dataset" that is similar in many points to the dataframes in R and can manage both: strings and numerical values (also including dates).

Import/Export *.csv files as data set (data frame)

Let us first see how to import directly as dataset using the function **dataset( )** data from as *.csv file:

```
MATLAB Command Window
» mydata=dataset('File','c:\tmp\Sales_03.csv','Delimiter',';');
» size(mydata)

ans =

   109    10

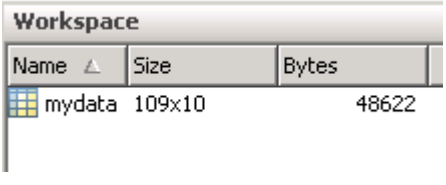
» mydata(1:10,:)

ans =

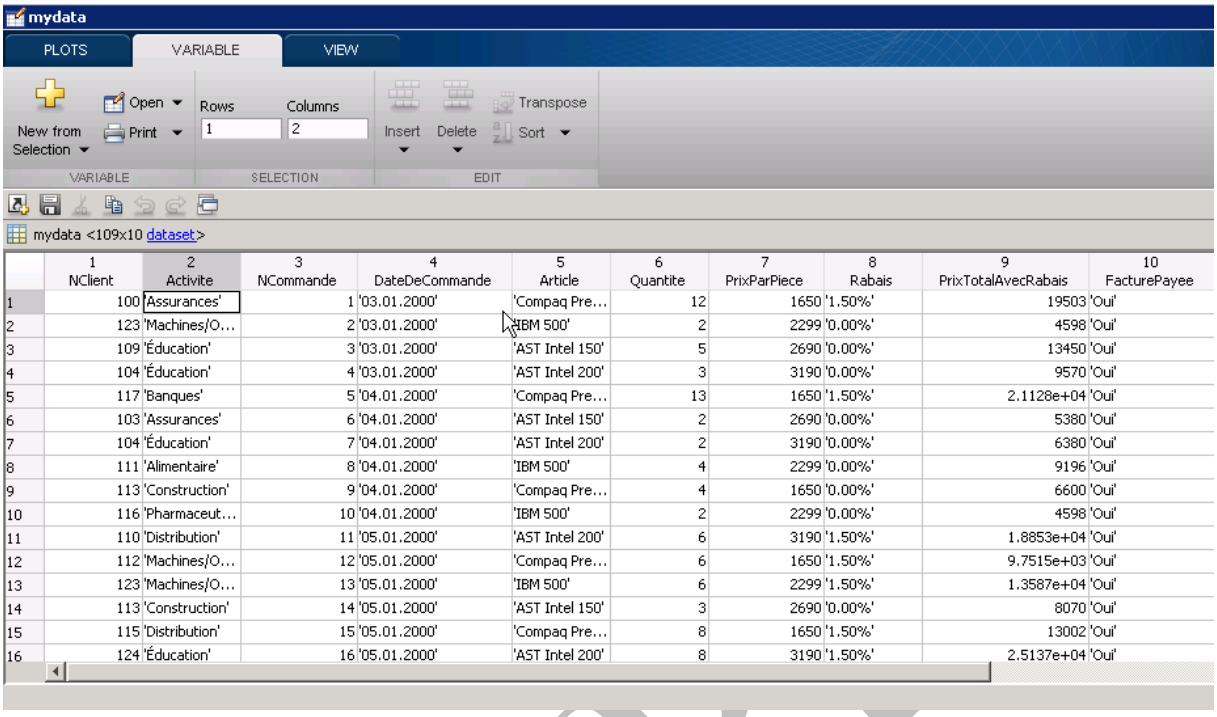
    NClient    Activite                NCommande    DateDeCommande      Article                  Quantite
    100        'Assurances'             1           '03.01.2000'        'Compaq Presario 100'    12
    123        'Machines/Outils'        2           '03.01.2000'        'IBM 500'                 2
    109        'Éducation'              3           '03.01.2000'        'AST Intel 150'           5
    104        'Éducation'              4           '03.01.2000'        'AST Intel 200'           3
    117        'Banques'                5           '04.01.2000'        'Compaq Presario 100'    13
    103        'Assurances'             6           '04.01.2000'        'AST Intel 150'           2
    104        'Éducation'              7           '04.01.2000'        'AST Intel 200'           2
    111        'Alimentaire'            8           '04.01.2000'        'IBM 500'                 4
    113        'Construction'           9           '04.01.2000'        'Compaq Presario 100'     4
    116        'Pharmaceutique'        10           '04.01.2000'        'IBM 500'                 2


    PrixParPiece    Rabais         PrixTotalAvecRabais    FacturePayee
    1650            '1.50%'        19503                  'Oui'
    2299            '0.00%'         4598                  'Oui'
    2690            '0.00%'        13450                  'Oui'
    3190            '0.00%'         9570                  'Oui'
```

And here this is how such a structure looks like in the variables workspace:

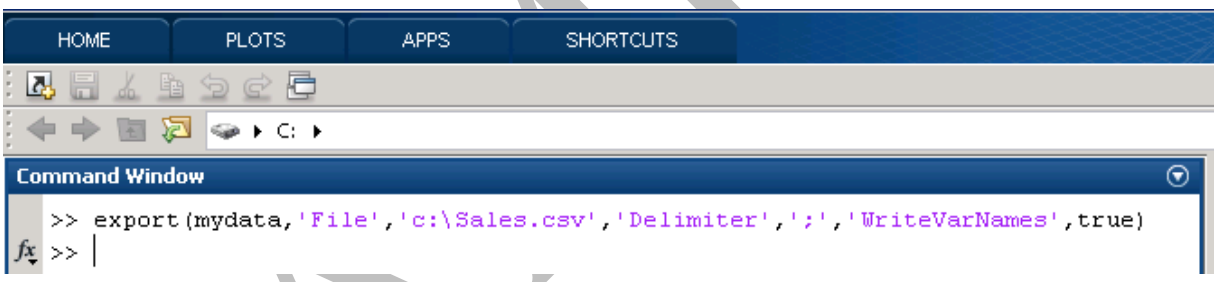


If you double click you can easily edit all data:



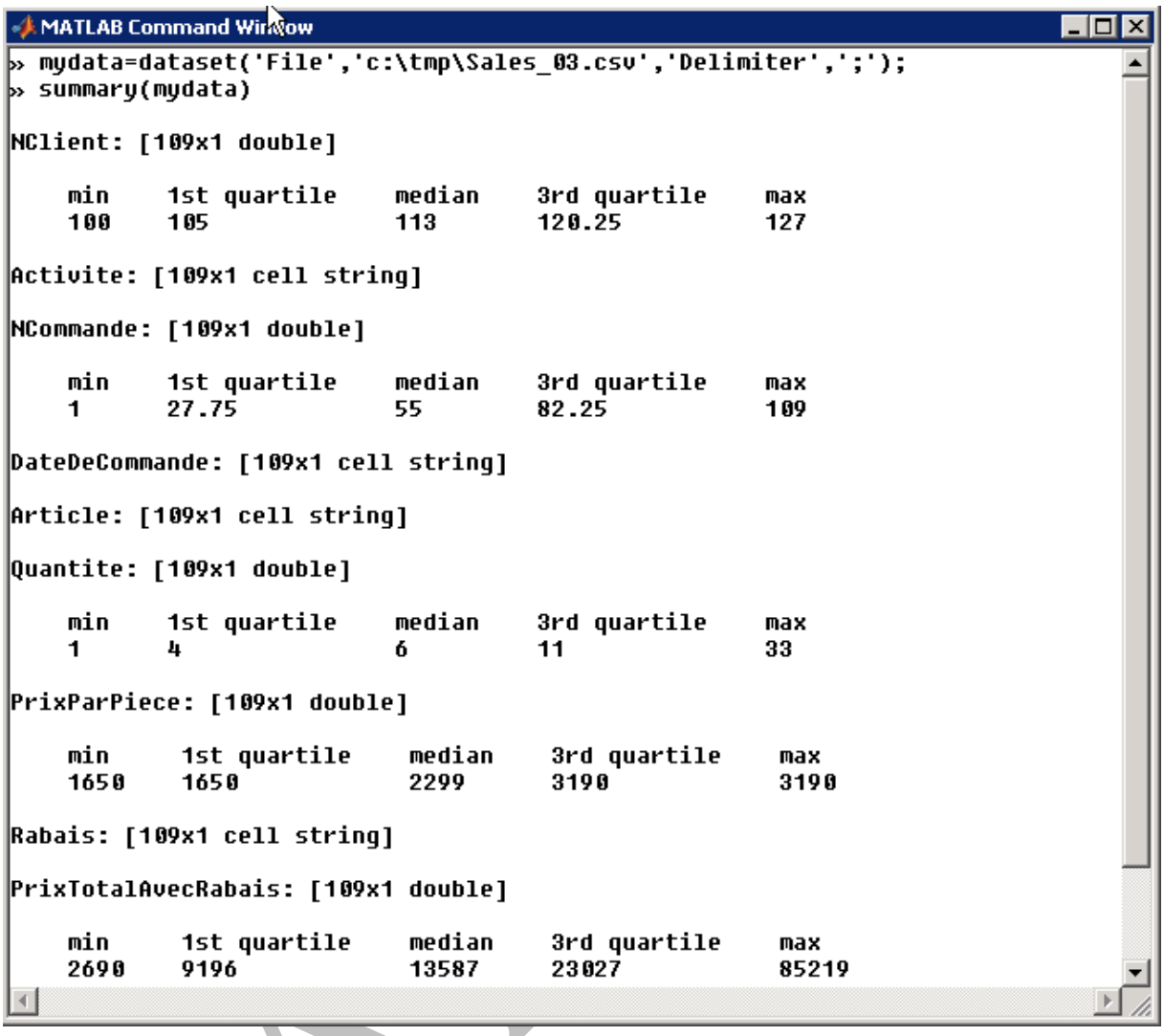
	1 NClient	2 Activite	3 NCommande	4 DateDeCommande	5 Article	6 Quantite	7 PrixParPiece	8 Rabais	9 PrixTotalAvecRabais	10 FacturePayee
1	100	'Assurances'	1	'03.01.2000'	'Compaq Pre...	12	1650	'1.50%'	19503	'Oui'
2	123	'Machines/O...	2	'03.01.2000'	'IBM 500'	2	2299	'0.00%'	4598	'Oui'
3	109	'Éducation'	3	'03.01.2000'	'AST Intel 150'	5	2690	'0.00%'	13450	'Oui'
4	104	'Éducation'	4	'03.01.2000'	'AST Intel 200'	3	3190	'0.00%'	9570	'Oui'
5	117	'Banques'	5	'04.01.2000'	'Compaq Pre...	13	1650	'1.50%'	2.1128e+04	'Oui'
6	103	'Assurances'	6	'04.01.2000'	'AST Intel 150'	2	2690	'0.00%'	5380	'Oui'
7	104	'Éducation'	7	'04.01.2000'	'AST Intel 200'	2	3190	'0.00%'	6380	'Oui'
8	111	'Alimentaire'	8	'04.01.2000'	'IBM 500'	4	2299	'0.00%'	9196	'Oui'
9	113	'Construction'	9	'04.01.2000'	'Compaq Pre...	4	1650	'0.00%'	6600	'Oui'
10	116	'Pharmaceut...	10	'04.01.2000'	'IBM 500'	2	2299	'0.00%'	4598	'Oui'
11	110	'Distribution'	11	'05.01.2000'	'AST Intel 200'	6	3190	'1.50%'	1.8853e+04	'Oui'
12	112	'Machines/O...	12	'05.01.2000'	'Compaq Pre...	6	1650	'1.50%'	9.7515e+03	'Oui'
13	123	'Machines/O...	13	'05.01.2000'	'IBM 500'	6	2299	'1.50%'	1.3587e+04	'Oui'
14	113	'Construction'	14	'05.01.2000'	'AST Intel 150'	3	2690	'0.00%'	8070	'Oui'
15	115	'Distribution'	15	'05.01.2000'	'Compaq Pre...	8	1650	'1.50%'	13002	'Oui'
16	124	'Éducation'	16	'05.01.2000'	'AST Intel 200'	8	3190	'1.50%'	2.5137e+04	'Oui'

To export a dataset to a *.csv file you just have to use the function **export( )**:



SAMPLE

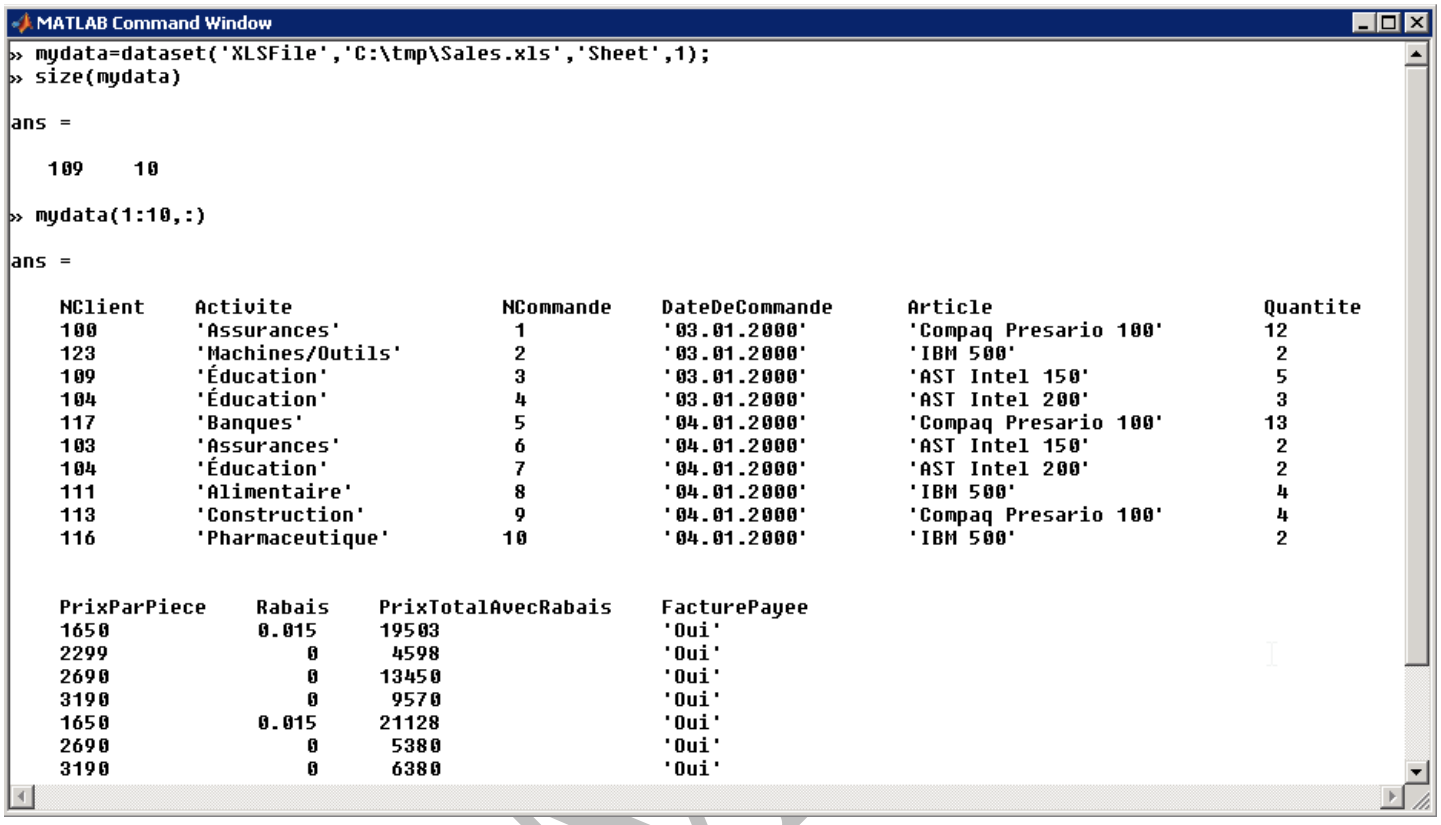
Like in R you can get a summary of the dataset using the function **summary( )**:



SA

Import *.xls/*.xlsx files as data set (data frame)

Let us now see how to import directly as dataset using the function **dataset( )** data from as *.xls or *.xlsx file (Microsoft Excel):



```
MATLAB Command Window
» mydata=dataset('XLSFile','C:\tmp\Sales.xls','Sheet',1);
» size(mydata)

ans =

   109    10

» mydata(1:10,:)

ans =

    NClient    Activite                NCommande    DateDeCommande      Article                 Quantite
    100        'Assurances'             1           '03.01.2000'        'Compaq Presario 100'   12
    123        'Machines/Outils'        2           '03.01.2000'        'IBM 500'                2
    109        'Éducation'              3           '03.01.2000'        'AST Intel 150'          5
    104        'Éducation'              4           '03.01.2000'        'AST Intel 200'          3
    117        'Banques'                5           '04.01.2000'        'Compaq Presario 100'   13
    103        'Assurances'             6           '04.01.2000'        'AST Intel 150'          2
    104        'Éducation'              7           '04.01.2000'        'AST Intel 200'          2
    111        'Alimentaire'            8           '04.01.2000'        'IBM 500'                4
    113        'Construction'           9           '04.01.2000'        'Compaq Presario 100'    4
    116        'Pharmaceutique'        10           '04.01.2000'        'IBM 500'                2


    PrixParPiece    Rabais    PrixTotalAvecRabais    FacturePayee
    1650            0.015     19503                  'Oui'
    2299                0      4598                  'Oui'
    2690                0     13450                  'Oui'
    3190                0      9570                  'Oui'
    1650            0.015     21128                  'Oui'
    2690                0      5380                  'Oui'
    3190                0      6380                  'Oui'
```

This works well!

SAM

Convert structure array to data sets

Sometimes (especially for databases) you cannot (as far as I know) directly import as dataset structure. Then it is useful to know how to convert a cell array to a dataset.

For this purpose we need to loop across the columns (**ToDo**: find an easier way to do that):

```
clear all;clc
formatSpec = '%s%s%s%s%s%s%s%s%s%s%s';
fileID = fopen('C:\tmp\Sales_03.csv');

dataArray = textscan(fileID, formatSpec,'Delimiter',';');
fclose(fileID);

%we create a cell array of texts only of the same dimensions as data array
%(minus the title for the rows) to convert everything as string
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)
    raw(:,col) = dataArray{col};
end
mydata=cell2dataset(raw);
```

The result will be almost as expected!:

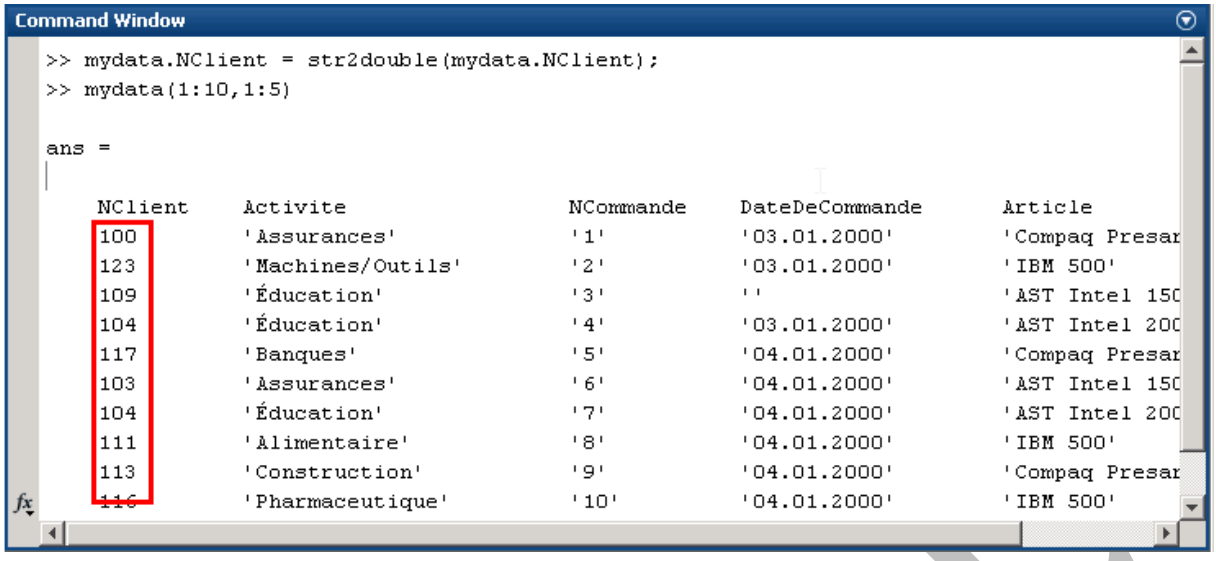
```
mydata =

    NClient       Activite                NCommande    DateDeCommande
    '100'         'Assurances'            '1'          '03.01.2000'
    '123'         'Machines/Outils'       '2'          '03.01.2000'
    '109'         'Éducation'             '3'          '03.01.2000'
    '104'         'Éducation'             '4'          '03.01.2000'
    '117'         'Banques'               '5'          '04.01.2000'
    '103'         'Assurances'            '6'          '04.01.2000'
    '104'         'Éducation'             '7'          '04.01.2000'
    '111'         'Alimentaire'           '8'          '04.01.2000'
    '113'         'Construction'          '9'          '04.01.2000'
    '116'         'Pharmaceutique'        '10'         '04.01.2000'
    '110'         'Distribution'          '11'         '05.01.2000'
    '112'         'Machines/Outils'       '12'         '05.01.2000'
    '123'         'Machines/Outils'       '13'         '05.01.2000'
    '113'         'Construction'          '14'         '05.01.2000'
    '115'         'Distribution'          '15'         '05.01.2000'
    '124'         'Éducation'             '16'         '05.01.2000'
    '124'         'Éducation'             '17'         '05.01.2000'
```

Workspace

Name	Size	Bytes
ans	1x1	8
col	1x1	8
dataArray	1x10	79658
fileID	1x1	8
formatSpec	1x20	40
mydata	109x10	80520
raw	110x10	79058

But we still have to convert the numeric string values to real pure numeric values with the function **str2double( )**:

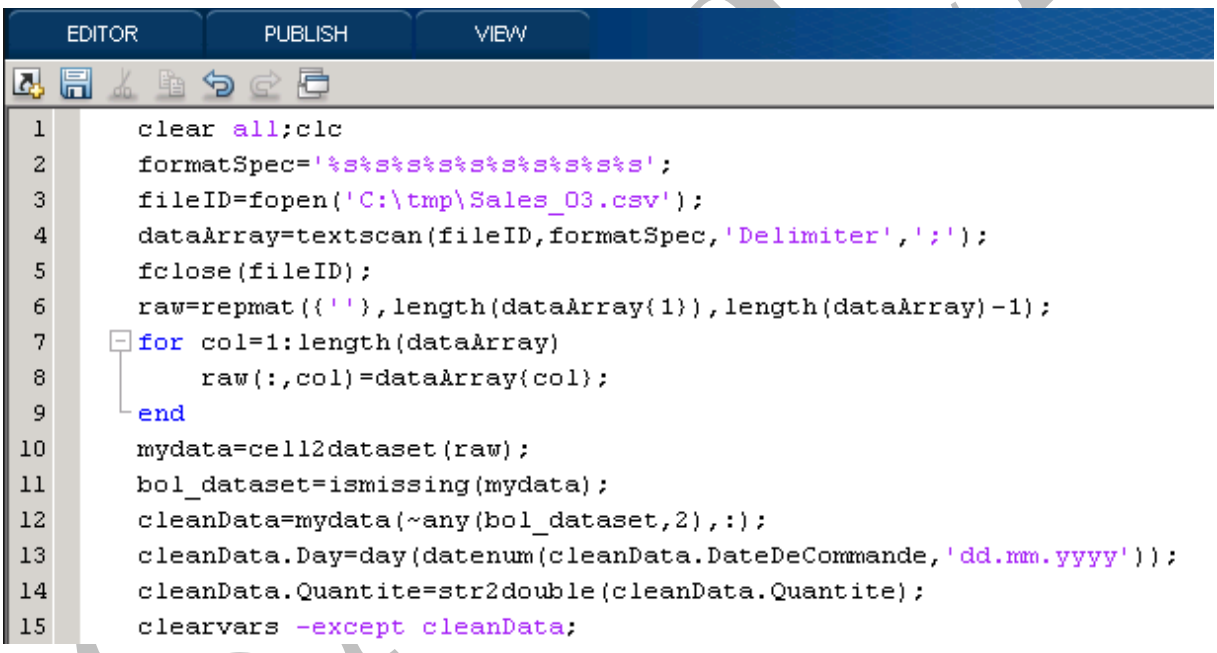


```
Command Window
>> mydata.NClient = str2double(mydata.NClient);
>> mydata(1:10,1:5)

ans =

    NClient    Activite                NCommande    DateDeCommande      Article
    100        'Assurances'            '1'          '03.01.2000'        'Compaq Presar
    123        'Machines/Outils'       '2'          '03.01.2000'        'IBM 500'
    109        'Éducation'             '3'          ''                  'AST Intel 150
    104        'Éducation'             '4'          '03.01.2000'        'AST Intel 200
    117        'Banques'               '5'          '04.01.2000'        'Compaq Presar
    103        'Assurances'            '6'          '04.01.2000'        'AST Intel 150
    104        'Éducation'             '7'          '04.01.2000'        'AST Intel 200
    111        'Alimentaire'           '8'          '04.01.2000'        'IBM 500'
    113        'Construction'          '9'          '04.01.2000'        'Compaq Presar
    116        'Pharmaceutique'        '10'         '04.01.2000'        'IBM 500'
```

Then you can do as script of this:



```
EDITOR  PUBLISH  VIEW

1   clear all;clc
2   formatSpec='%s%s%s%s%s%s%s%s%s%s';
3   fileID=fopen('C:\tmp\Sales_03.csv');
4   dataArray=textscan(fileID,formatSpec,'Delimiter',';');
5   fclose(fileID);
6   raw=repmat({''},length(dataArray{1}),length(dataArray)-1);
7   for col=1:length(dataArray)
8       raw(:,col)=dataArray{col};
9   end
10  mydata=cell2dataset(raw);
11  bol_dataset=ismissing(mydata);
12  cleanData=mydata(~any(bol_dataset,2),:);
13  cleanData.Day=day(datenum(cleanData.DateDeCommande,'dd.mm.yyyy'));
14  cleanData.Quantite=str2double(cleanData.Quantite);
15  clearvars -except cleanData;
```

Clean messy or missing data in datasets

Cleanings messing or missing datas is a major topic in finance and there are a lot of technics for this job. One of the most basic one is just to remove the problematic rows.

For the example consider the following dataset:

```
MATLAB Command Window
» mydata=dataset('File','c:\tmp\SalesToClean.csv','Delimiter',';');
» mydata(1:10,1:5)

ans =

    NClient    Activite                NCommande    DateDeCommande      Article
    100        'Assurances'             1           '03.01.2000'        'Compaq Presario 100'
    123        'Machines/Outils'        2           '03.01.2000'        'IBM 500'
    109        'Éducation'              3           ''                  'AST Intel 150'
    104        'Éducation'              4           '03.01.2000'        'AST Intel 200'
    117        'Banques'                5           '04.01.2000'        'Compaq Presario 100'
    103        'Assurances'             6           '04.01.2000'        'AST Intel 150'
    104        'Éducation'              7           '04.01.2000'        'AST Intel 200'
    111        'Alimentaire'            8           '04.01.2000'        'IBM 500'
    113        'Construction'           9           '04.01.2000'        'Compaq Presario 100'
    116        'Pharmaceutique'        10           '04.01.2000'        'IBM 500'

» |
```

To remove the missing data rows we will first use the **ismissing( )** function:

```
MATLAB Command Window
» mydata=dataset('File','c:\tmp\SalesToClean.csv','Delimiter',';');
» bol_dataset=ismissing(mydata)

bol_dataset =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

And after we use the **any( )** function to extract only problematic rows or to eliminate problematic rows:

```
MATLAB Command Window
» messyData=mydata(any(bol_dataset,2),:)

messyData =

    NClient    Activite           NCommande    DateDeCommande    Article                Quantite
    109        'Éducation'        3            ''                'AST Intel 150'        5


    PrixParPiece    Rabais         PrixTotalAvecRabais    FacturePayee
    2690            '0.00%'        13450                  'Oui'

» cleanData=mydata(~any(bol_dataset,2),:);
» cleanData(1:10,1:5)

ans =

    NClient    Activite                NCommande    DateDeCommande      Article
    100        'Assurances'             1           '03.01.2000'        'Compaq Presario 100'
    123        'Machines/Outils'        2           '03.01.2000'        'IBM 500'
    104        'Éducation'              4           '03.01.2000'        'AST Intel 200'
    117        'Banques'                5           '04.01.2000'        'Compaq Presario 100'
    103        'Assurances'             6           '04.01.2000'        'AST Intel 150'
    104        'Éducation'              7           '04.01.2000'        'AST Intel 200'
    111        'Alimentaire'            8           '04.01.2000'        'IBM 500'
    113        'Construction'           9           '04.01.2000'        'Compaq Presario 100'
    116        'Pharmaceutique'        10           '04.01.2000'        'IBM 500'
    110        'Distribution'          11           '05.01.2000'        'AST Intel 200'

»
```

Another typical cleaning process is to reformate dates. Here is the beginning of such an example (take care! to use the **month( )** function you must have the Finance Toolbox for recall!!!!):

```
Command Window
>> cleanData.Month=month(datenum(cleanData.DateDeCommande,'dd.mm.yyyy'));
>> cleanData(1:3,:)

ans =

    NClient      Activite                NCommande    DateDeCommande
    '100'        'Assurances'            '1'          '03.01.2000'
    '123'        'Machines/Outils'       '2'          '03.01.2000'
    '104'        'Éducation'             '4'          '03.01.2000'

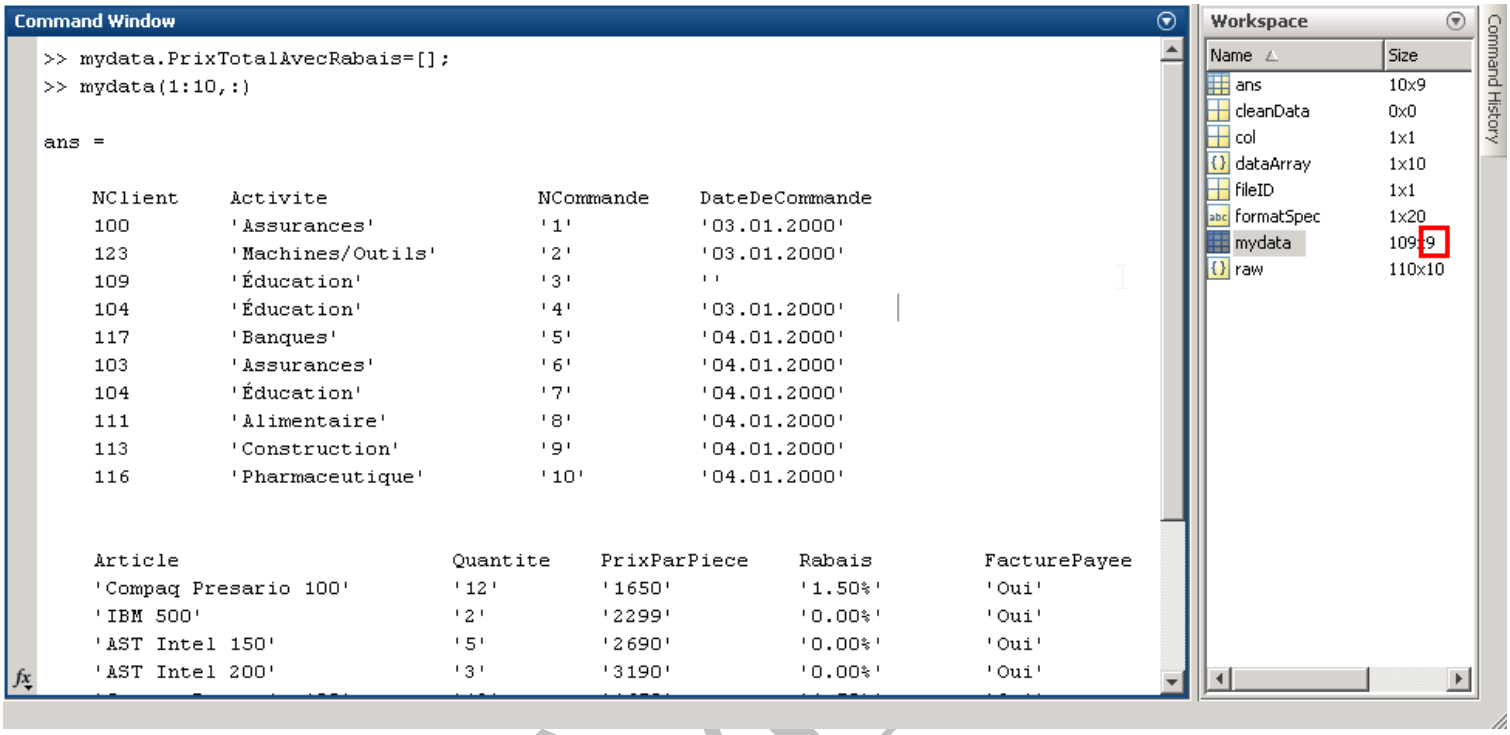

    Article                    Quantite    PrixParPiece    Rabais
    'Compaq Presario 100'      '12'        '1650'          '1.50%'
    'IBM 500'                  '2'         '2299'          '0.00%'
    'AST Intel 200'            '3'         '3190'          '0.00%'


    PrixTotalAvecRabais    FacturePayee    Month
    '19503'                'Oui'           1
    '4598'                 'Oui'           1
    '9570'                 'Oui'           1

fx >>
```

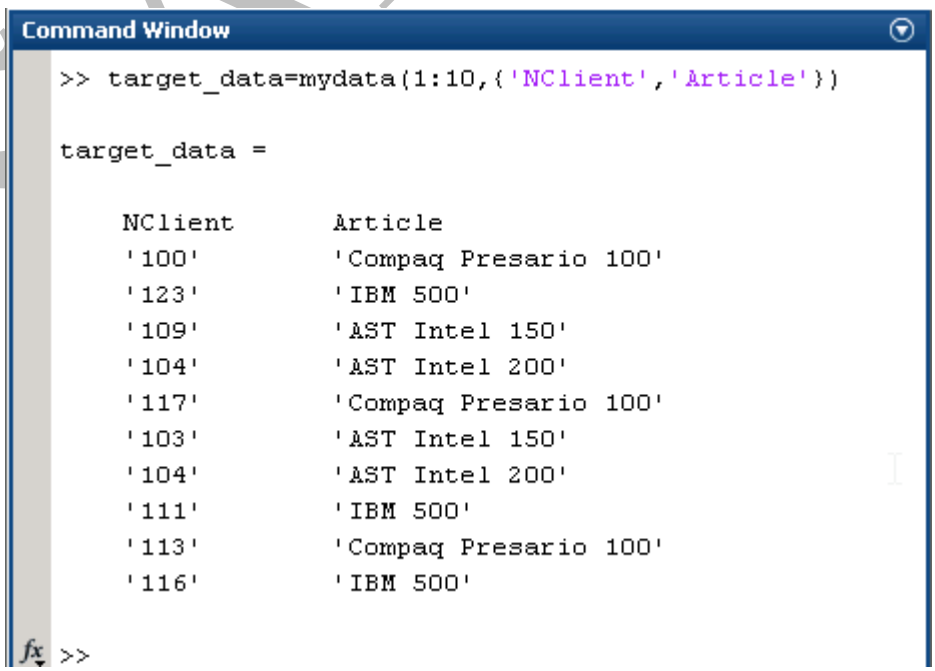
Select/Remove column (variable)

One very common question about dataset is how it is possible to remove on column (variable). For this consider we want to remove the column *PrixTotalAvecRabais*. The syntax will be the following:



Create subsets and filter

Another very common question is how to select a subset of rows and columns of a dataset. For this the syntax is simply the following:



Or if you need to create a subset trough a filter, the syntax is the following for numerical values:

```
Command Window
>> target_data=mydata(mydata.Quantite==10,:)

target_data =

    NClient      Activite              NCommande    DateCommande
    '118'        'Éducation'           '61'         '18.01.2000'
    '101'        'Construction'        '76'         '21.01.2000'


    Article               Quantite    PrixParPiece    Rabais
    'IBM 500'             10          2299            '1.50%'
    'AST Intel 200'       10          3190            '1.50%'


    FacturePayee    PrixTotal
    'Oui'           22990
    'Non'           31900

fx >>
```

For strings this is a little bit more complicate (**ToDo**: found if there is not a more simple way to do this job):

```
Command Window
>> mydata(find(strncmp(mydata.Activite,'Banques',length('Banques'))),3:end)

ans =

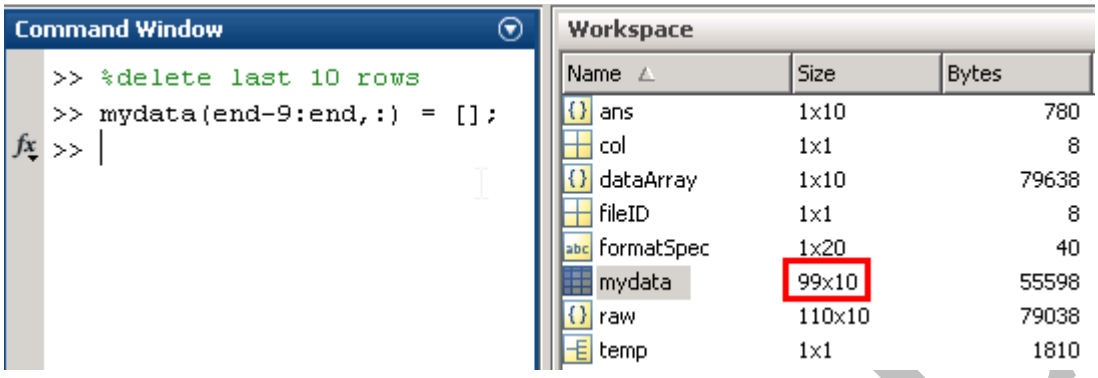
    NCommande    DateCommande        Article                    Quantite
    '5'          '04.01.2000'        'Compaq Presario 100'      13
    '31'         '10.01.2000'        'AST Intel 150'             2
    '54'         '17.01.2000'        'IBM 500'                   5
    '59'         '18.01.2000'        'IBM 500'                   9
    '67'         '19.01.2000'        'IBM 500'                  12
    '68'         '19.01.2000'        'AST Intel 150'             5
    '86'         '25.01.2000'        'Compaq Presario 100'      19
    '87'         '25.01.2000'        'IBM 500'                   6


    PrixParPiece    Rabais          FacturePayee    PrixTotal
    1650            '1.50%'         'Oui'           21450
    2690            '0.00%'         'Oui'            5380
    2299            '0.00%'         'Oui'           11495
    2299            '1.50%'         'Oui'           20691
    2299            '1.50%'         'Non'           27588
    2690            '0.00%'         'Non'           13450
    1650            '3.00%'         'Non'           31350
    2299            '1.50%'         'Non'           13794

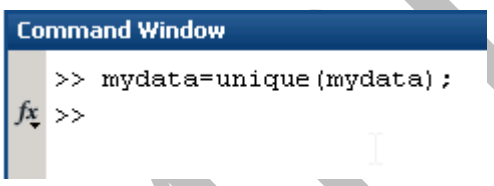
fx
```

Delete rows and clean duplicates

Delete rows in a dataset is exactly similar to delete rows in a matrix or vector. The syntax is just the following:

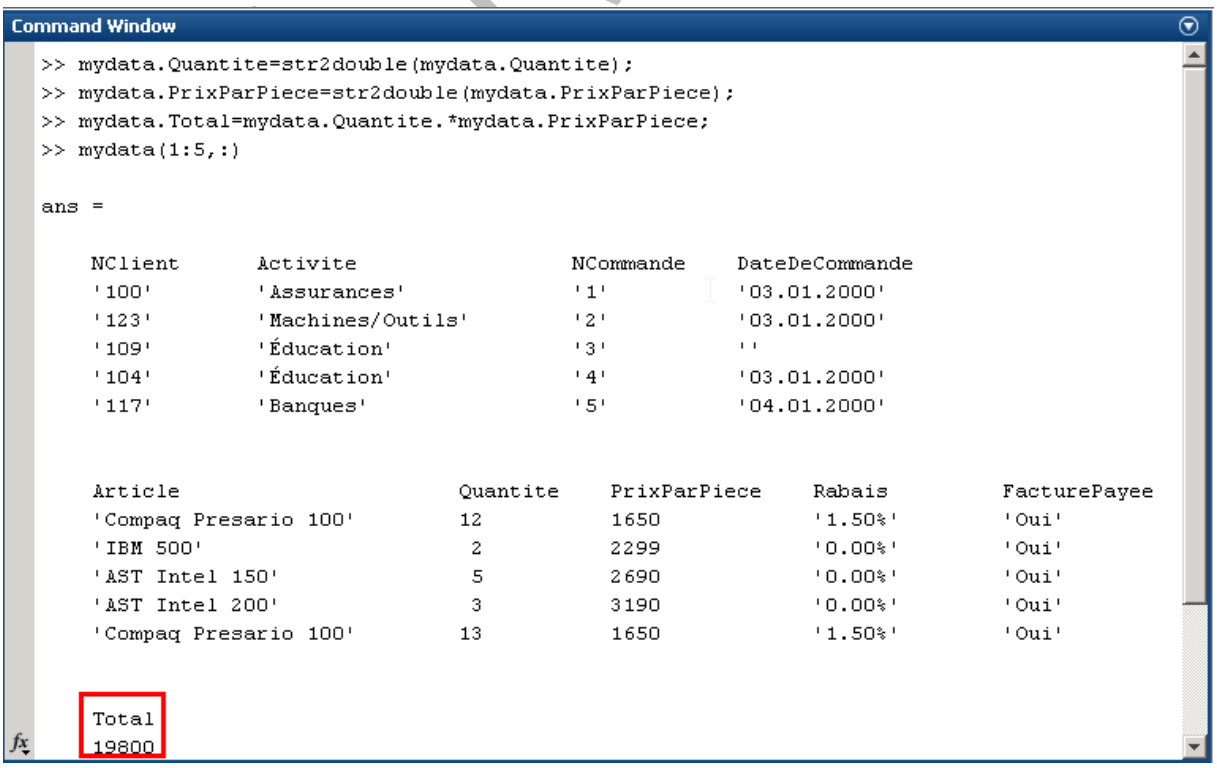


The function to clean the duplicates of a dataset is the same as for cleaning duplicates in a vector or matrix. It use the **unique( )** function:



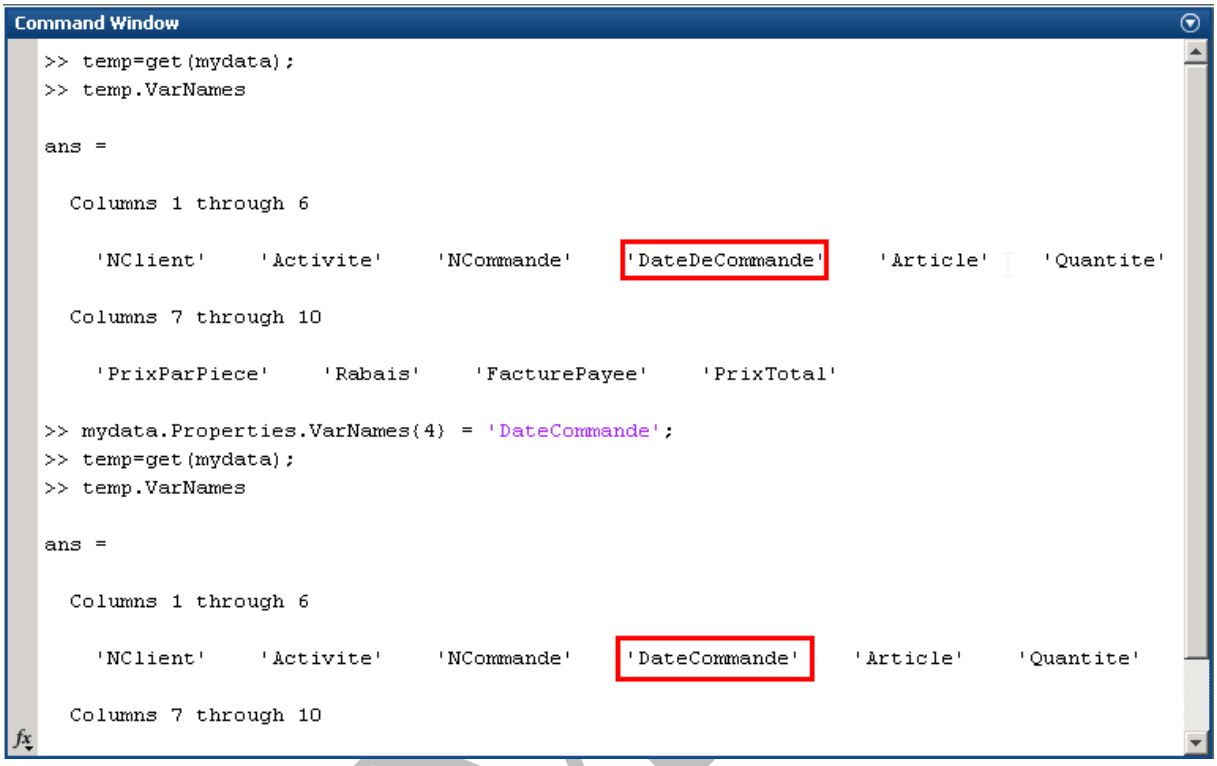
Add variables

One of the most common simple manipulation is to add new variable Consider the case where we want to add a new column whose results is the arithmetic operation of other columns:



Change variable name

Change a variable name could be very easy but the manipulation is a little bit strange to in MATLAB. We have get the old name before by using the property (or Method?) **VarNames** before we can be able to change the name.



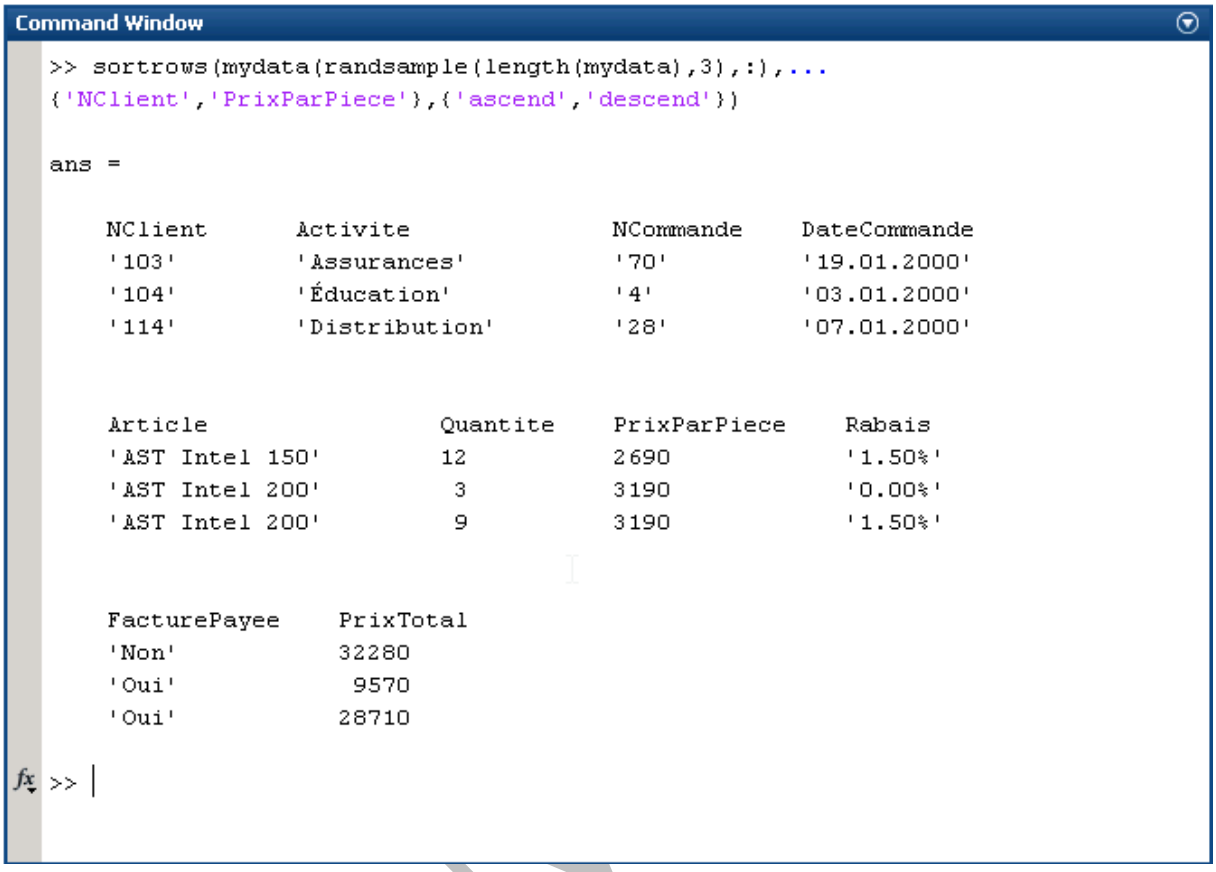
Change column order

ToDo... (search a better way than recreating a new dataset...)

SAMPLE

Sort a dataset

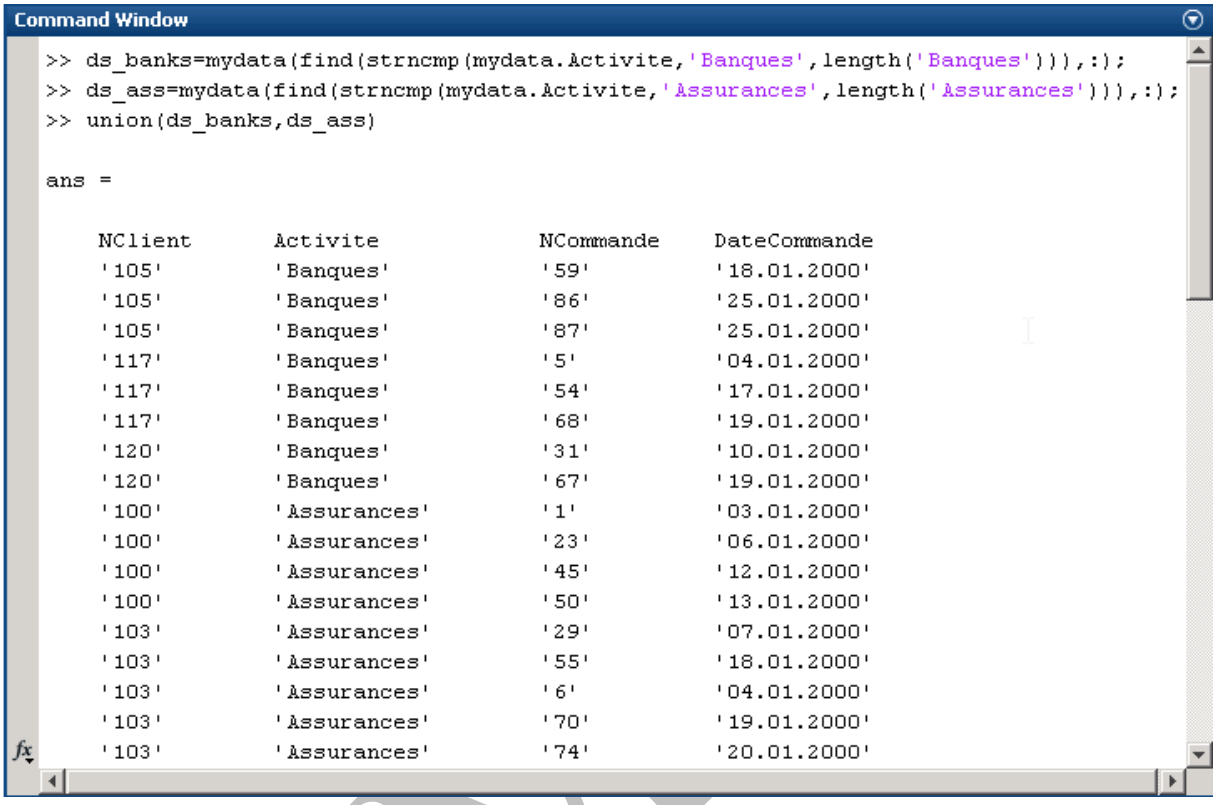
The command to sort a dataset in MATLAB is very simple. You just have to use the commande **sortrows( )**:



SAM

Merge datasets

Another typical is to be able to merge two datasets having the same number of columns and same order of columns. For this purpose we simply use the functions **union( )** as for vectors:

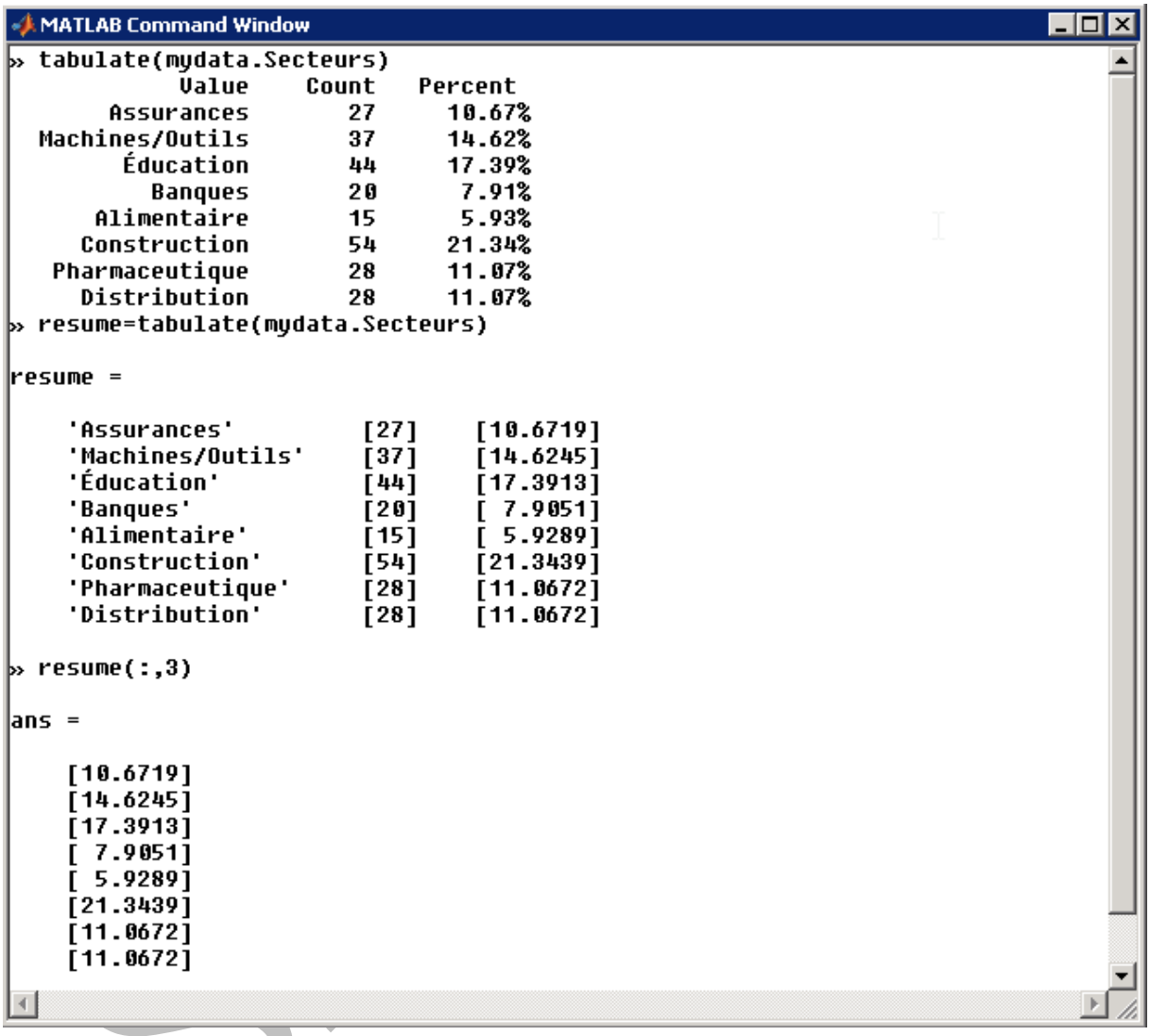


SAM

Dataset tabulations and crosstab analysis

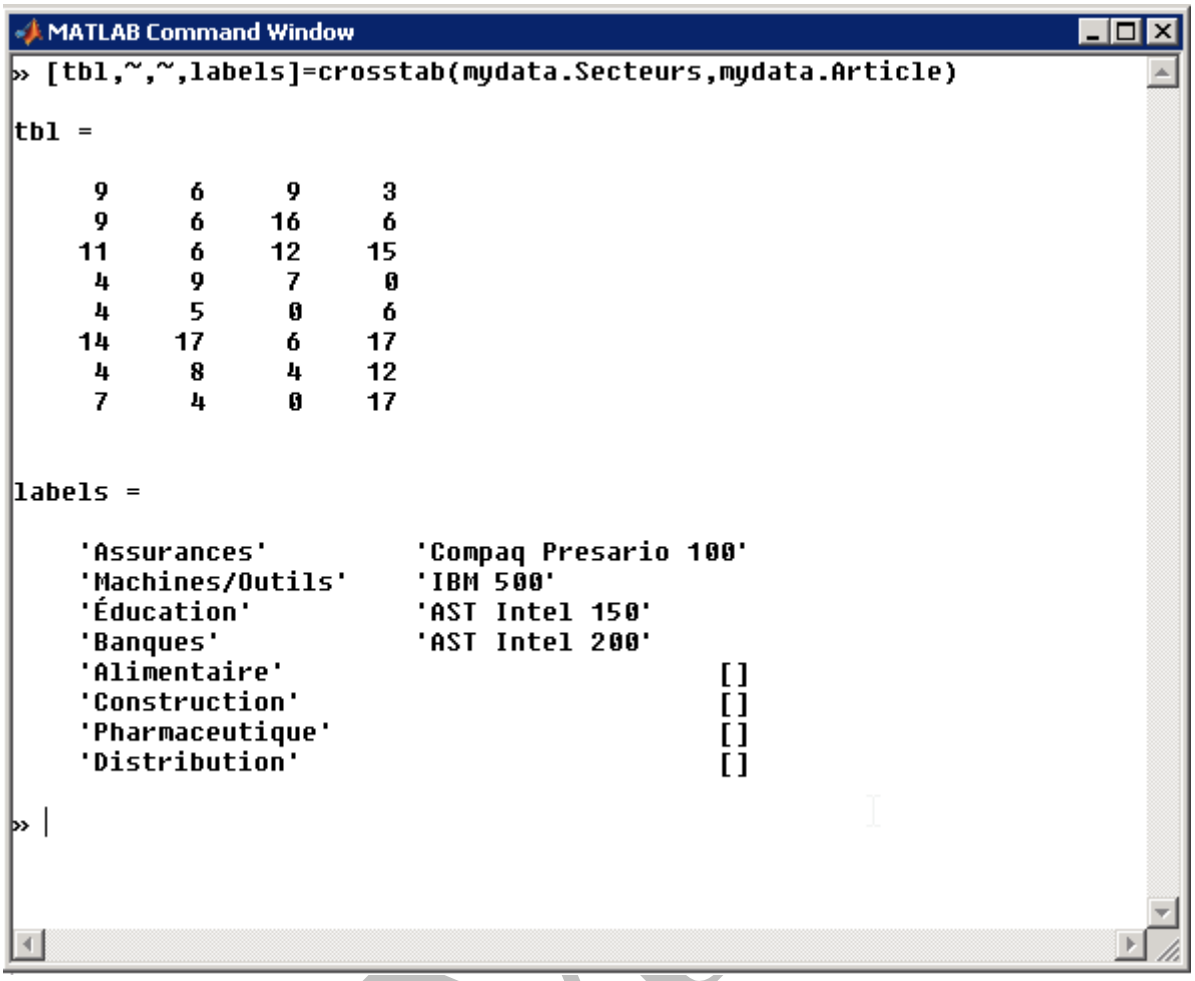
Frequencies analysis of a dataset is very useful in many applications for charting or statistical analysis (we will also see a very important application stratified random sampling further).

Here is an first overview of the **tabulate( )** function:



We get the same values as in R and Microsoft Excel… ouf!

And now and look to the **crosstab( )** function where we can see that for sure… we are far away from the flexibility of a spreadsheet software:

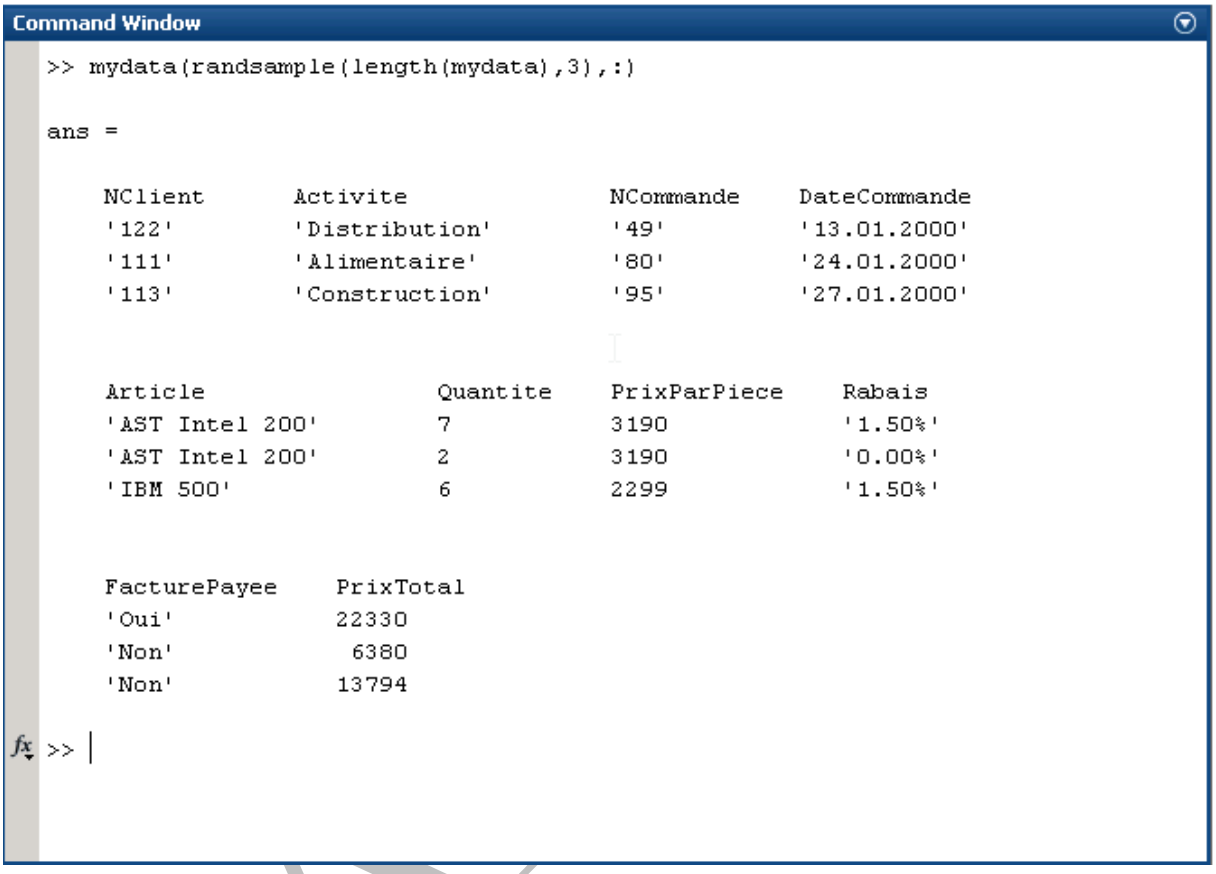

MATLAB Command Window
» [tbl,~,~,labels]=crosstab(mydata.Secteurs,mydata.Article)
tbl =
9 6 9 3
9 6 16 6
11 6 12 15
4 9 7 0
4 5 0 6
14 17 6 17
4 8 4 12
7 4 0 17
labels =
'Assurances' 'Compaq Presario 100'
'Machines/Outils' 'IBM 500'
'Éducation' 'AST Intel 150'
'Banques' 'AST Intel 200'
'Alimentaire' []
'Construction' []
'Pharmaceutique' []
'Distribution' []
»

SAM

Random sample of a dataset

Simple random sampling (equal probabilities)

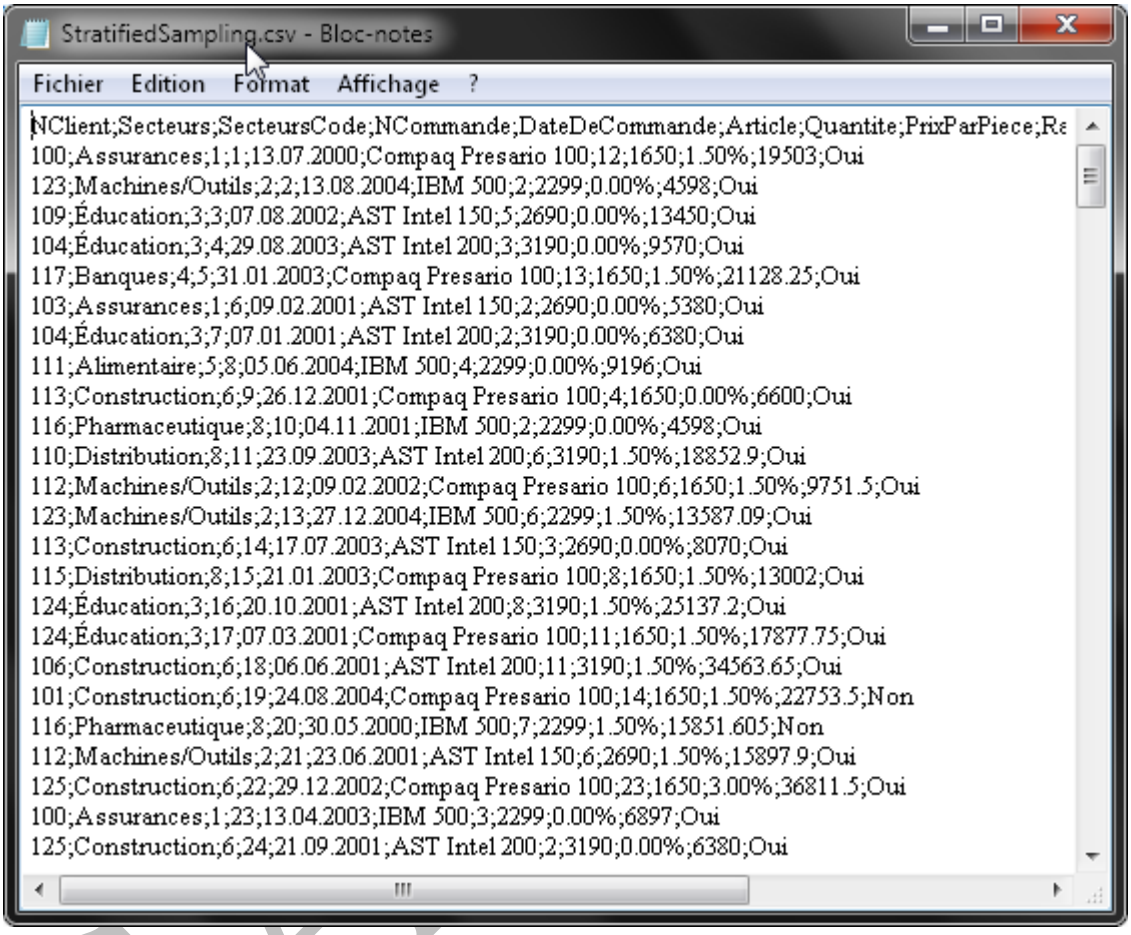
Sampling in quality control is an important step of statistical inference. For this purpose the method is the same as the one for vectors using the functions **randsample( )** and **length( )** together:



Balanced Stratified random sampling (unequal probabilities)

We know that the previous example did not respect any proportions of whatever is the content of the dataset. Now suppose we want to respect to have the same number of Activity Sectors columns (it very important sometimes in surveys analysis to have identical proportions).

For this purpose we will use the following dataset of 254 records:



We write the following script:

StratifiedSampli... ×

```
1  clc;clear all
2  mydata=dataset('File','c:\tmp\StratifiedSampling.csv','Delimiter',';');
3
4  fraction = 0.3; %Take care! Must not be to high for randsample()
5  observations=mydata.SecteursCode;
6
7  classes = unique(observations);
8  nObs = length(observations)
9  nClasses = length(classes)
10 nSamples = round(nObs * fraction / nClasses)
11
12 for ii = 1:nClasses
13     idx = observations == classes(ii);
14     filterData=mydata(idx,:);
15     if ii==1
16         samples=filterData(randsample(length(filterData), nSamples),:);
17     else
18         samples=union(filterData(randsample(length(filterData), nSamples),:),samples);
19     end
20 end
```

So we get:

Command Window

```
nObs =

   253


nClasses =

     7


nSamples =

    11

fx >>
```

And if we take a lot to the content of the *samples* variable:

samples <72x11 dataset>

	1 NClient	2 Secteurs	3 SecteursCode	4 NCommande	5 DateDeCommande	6 Article	7 Quantite	8 PrixParPiece	9 Rabais
1	111	'Alimentaire'	5	32	'31.07.2004'	'Compaq Pre...	16	1650	'3.00%'
2	111	'Alimentaire'	5	80	'06.04.2000'	'AST Intel 200'	2	3190	'0.00%'
3	127	'Alimentaire'	5	34	'03.07.2003'	'IBM 500'	7	2299	'1.50%'
4	127	'Alimentaire'	5	44	'27.08.2002'	'AST Intel 200'	2	3190	'0.00%'
5	127	'Alimentaire'	5	58	'31.08.2000'	'Compaq Pre...	7	1650	'1.50%'
6	127	'Alimentaire'	5	142	'03.07.2003'	'IBM 500'	7	2299	'1.50%'
7	127	'Alimentaire'	5	166	'31.08.2000'	'Compaq Pre...	7	1650	'1.50%'
8	127	'Alimentaire'	5	208	'27.08.2002'	'AST Intel 200'	2	3190	'0.00%'
9	127	'Alimentaire'	5	227	'27.08.2002'	'AST Intel 200'	2	3190	'0.00%'
10	100	'Assurances'	1	45	'10.10.2003'	'AST Intel 200'	4	3190	'0.00%'
11	100	'Assurances'	1	153	'10.10.2003'	'AST Intel 200'	4	3190	'0.00%'
12	100	'Assurances'	1	233	'10.03.2004'	'AST Intel 150'	13	2690	'1.50%'
13	103	'Assurances'	1	6	'09.02.2001'	'AST Intel 150'	2	2690	'0.00%'
14	103	'Assurances'	1	137	'10.12.2004'	'Compaq Pre...	6	1650	'1.50%'
15	103	'Assurances'	1	178	'06.10.2001'	'AST Intel 150'	12	2690	'1.50%'
16	103	'Assurances'	1	184	'24.09.2000'	'Compaq Pre...	8	1650	'1.50%'
17	103	'Assurances'	1	195	'06.10.2001'	'AST Intel 150'	12	2690	'1.50%'
18	103	'Assurances'	1	252	'10.12.2004'	'Compaq Pre...	6	1650	'1.50%'
19	105	'Banques'	4	59	'27.11.2004'	'IBM 500'	9	2299	'1.50%'
20	105	'Banques'	4	86	'12.08.2003'	'Compaq Pre...	19	1650	'3.00%'
21	105	'Banques'	4	167	'27.11.2004'	'IBM 500'	9	2299	'1.50%'

We can see that each *Sector* occurs equally!

SAMPLE

Stratified random sampling (unequal probabilities)

And now if we want a sample of a given size but respecting the proportion of each sector, we just have to modify a little bit the previous example:

StratifiedSampli...

```
clc;clear all
mydata=dataset('File','c:\tmp\StratifiedSampling.csv','Delimiter',';');

observations=mydata.SecteursCode;

classes = unique(observations);
resume=tabulate(observations);
nObs = length(observations);
nClasses = length(classes);
nSamples = 50;

for ii = 1:nClasses
    idx = observations == classes(ii);
    filterData=mydata(idx,:);
    StrataSamples=round((resume(ii,3)/100)*nSamples)
    if ii==1
        samples=filterData(randsample(length(filterData), StrataSamples),:);
    else
        samples=union(filterData(randsample(length(filterData), StrataSamples),:),samples);
    end
end
```

If we take a look to the *samples* variable:

samples <51x11 dataset>

	1 NClient	2 Secteurs	3 SecteursCode	4 NCommande	5 DateDeCommande	6 Article	7 Quantite	8 PrixParPiece	9 Rabais	PrixTot
1	127	'Alimentaire'	5	93	'16.04.2000'	'AST Intel 200'	8	3190	'1.50%'	
2	127	'Alimentaire'	5	166	'31.08.2000'	'Compaq Pre...	7	1650	'1.50%'	
3	127	'Alimentaire'	5	208	'27.08.2002'	'AST Intel 200'	2	3190	'0.00%'	
4	100	'Assurances'	1	50	'10.03.2004'	'AST Intel 150'	13	2690	'1.50%'	
5	100	'Assurances'	1	233	'10.03.2004'	'AST Intel 150'	13	2690	'1.50%'	
6	103	'Assurances'	1	6	'09.02.2001'	'AST Intel 150'	2	2690	'0.00%'	
7	103	'Assurances'	1	178	'06.10.2001'	'AST Intel 150'	12	2690	'1.50%'	
8	103	'Assurances'	1	237	'24.09.2000'	'Compaq Pre...	8	1650	'1.50%'	
9	117	'Banques'	4	54	'18.06.2001'	'IBM 500'	5	2299	'0.00%'	
10	117	'Banques'	4	68	'17.02.2003'	'AST Intel 150'	5	2690	'0.00%'	
11	117	'Banques'	4	105	'08.06.2002'	'AST Intel 150'	3	2690	'0.00%'	
12	117	'Banques'	4	113	'31.01.2003'	'Compaq Pre...	13	1650	'1.50%'	
13	101	'Construction'	6	69	'12.10.2002'	'Compaq Pre...	13	1650	'1.50%'	
14	101	'Construction'	6	71	'18.04.2000'	'IBM 500'	5	2299	'0.00%'	
15	101	'Construction'	6	149	'05.08.2002'	'AST Intel 200'	1	3190	'0.00%'	
16	101	'Construction'	6	177	'12.10.2002'	'Compaq Pre...	13	1650	'1.50%'	
17	101	'Construction'	6	205	'05.08.2002'	'AST Intel 200'	1	3190	'0.00%'	
18	106	'Construction'	6	18	'06.06.2001'	'AST Intel 200'	11	3190	'1.50%'	
19	106	'Construction'	6	146	'09.01.2003'	'IBM 500'	4	2299	'0.00%'	
20	113	'Construction'	6	155	'28.08.2000'	'IBM 500'	11	2299	'1.50%'	
21	125	'Construction'	6	24	'21.09.2001'	'AST Intel 200'	2	3190	'0.00%'	

We can see that proportions are preserved and the results are the same as with the R software.

Plots

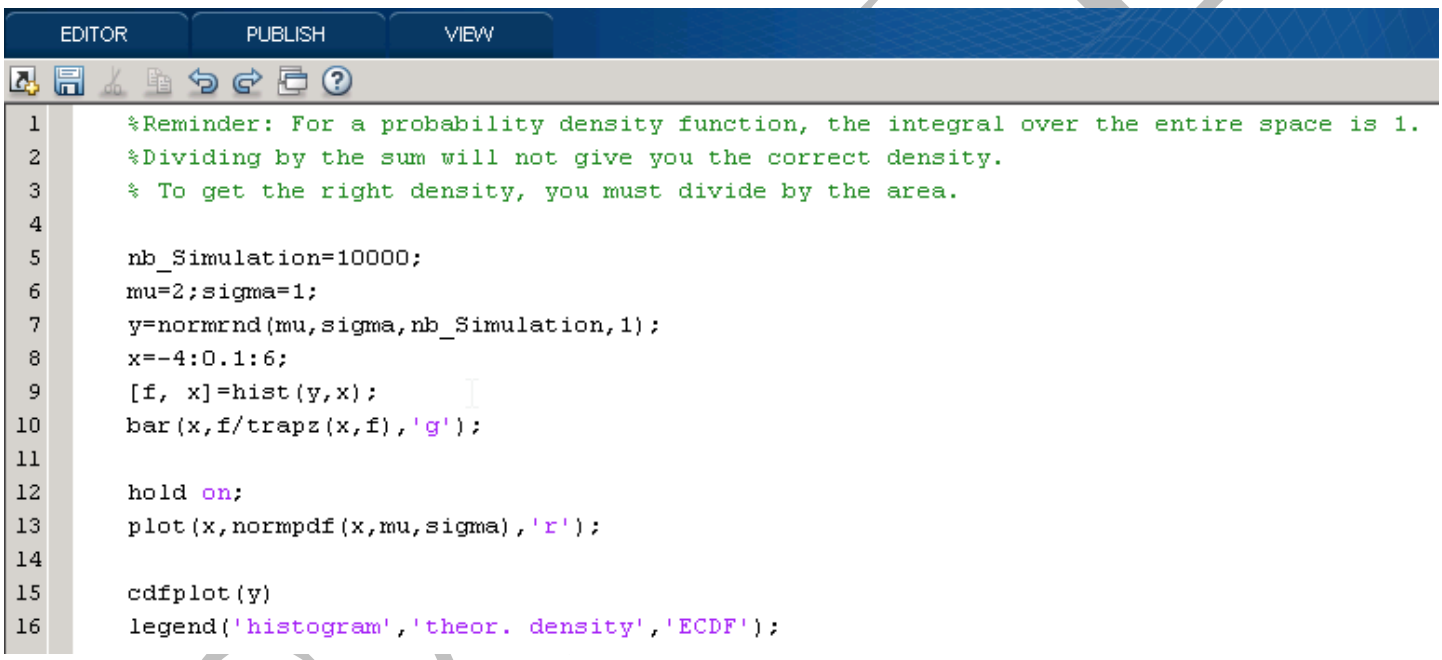
We have study during a very (too much) long time possible charts using only the core version of MATLAB (see page 262).

In this chapter we will focus on charts that can be done especially with the Statistical Toolbox.

Histogram/CDF/PDF

Create histograms is a very common task asked by many statistical softwares users even if it is not the best way to analyze the shape of data because of the bias of bin width.

The first method used the combination of functions **hist( )**, **bar( )**, **trapz( )** and **cdfplot( )** to plot on the same chart a histogram, a CDF and a PDF:



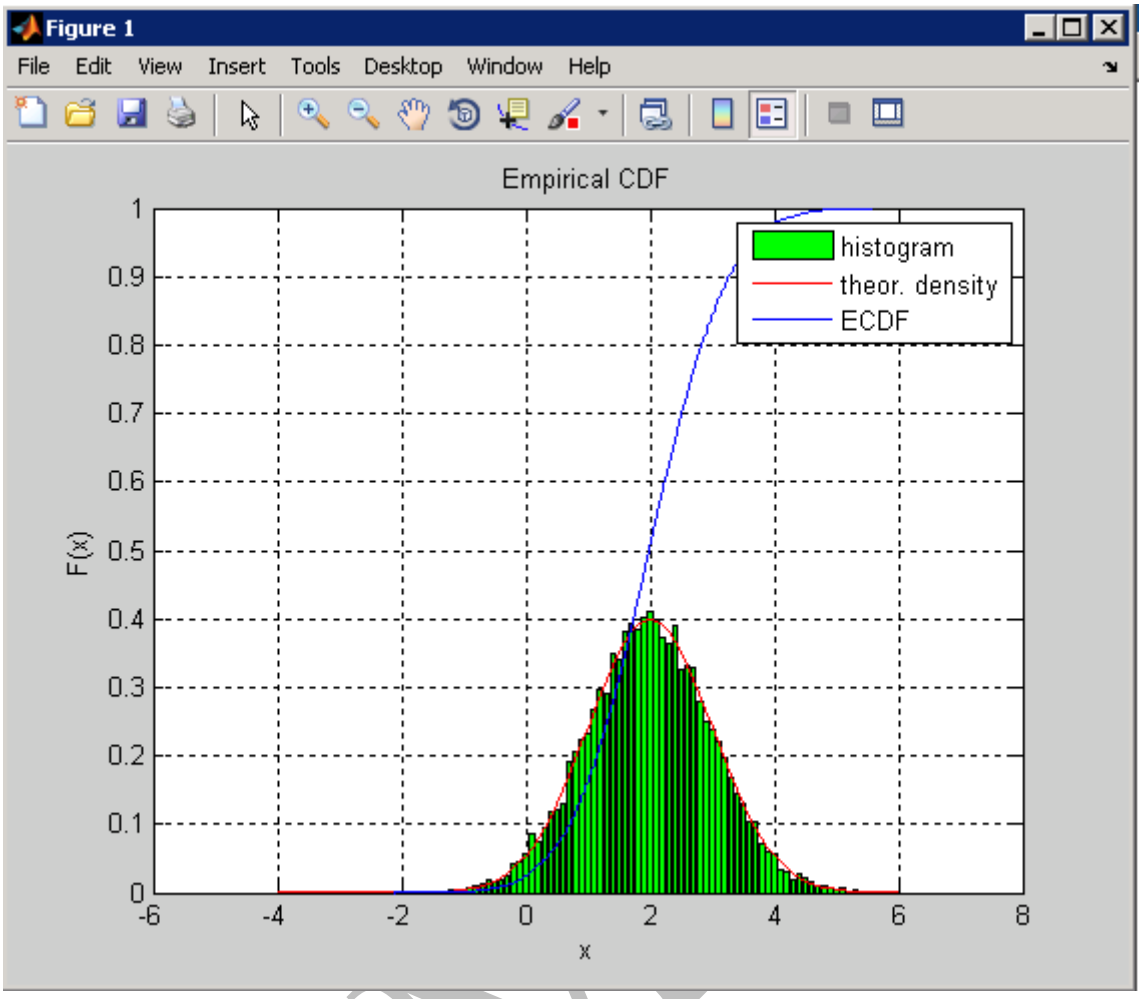

```
%Reminder: For a probability density function, the integral over the entire space is 1.
%Dividing by the sum will not give you the correct density.
% To get the right density, you must divide by the area.

nb_Simulation=10000;
mu=2;sigma=1;
y=normrnd(mu,sigma,nb_Simulation,1);
x=-4:0.1:6;
[f, x]=hist(y,x);
bar(x,f/trapz(x,f),'g');

hold on;
plot(x,normpdf(x,mu,sigma),'r');

cdfplot(y)
legend('histogram','theor. density','ECDF');
```

The result is visible on the next page.



The second method is a variant using **ecdf( )** instead of using a plot of the function with **cdf( )**:

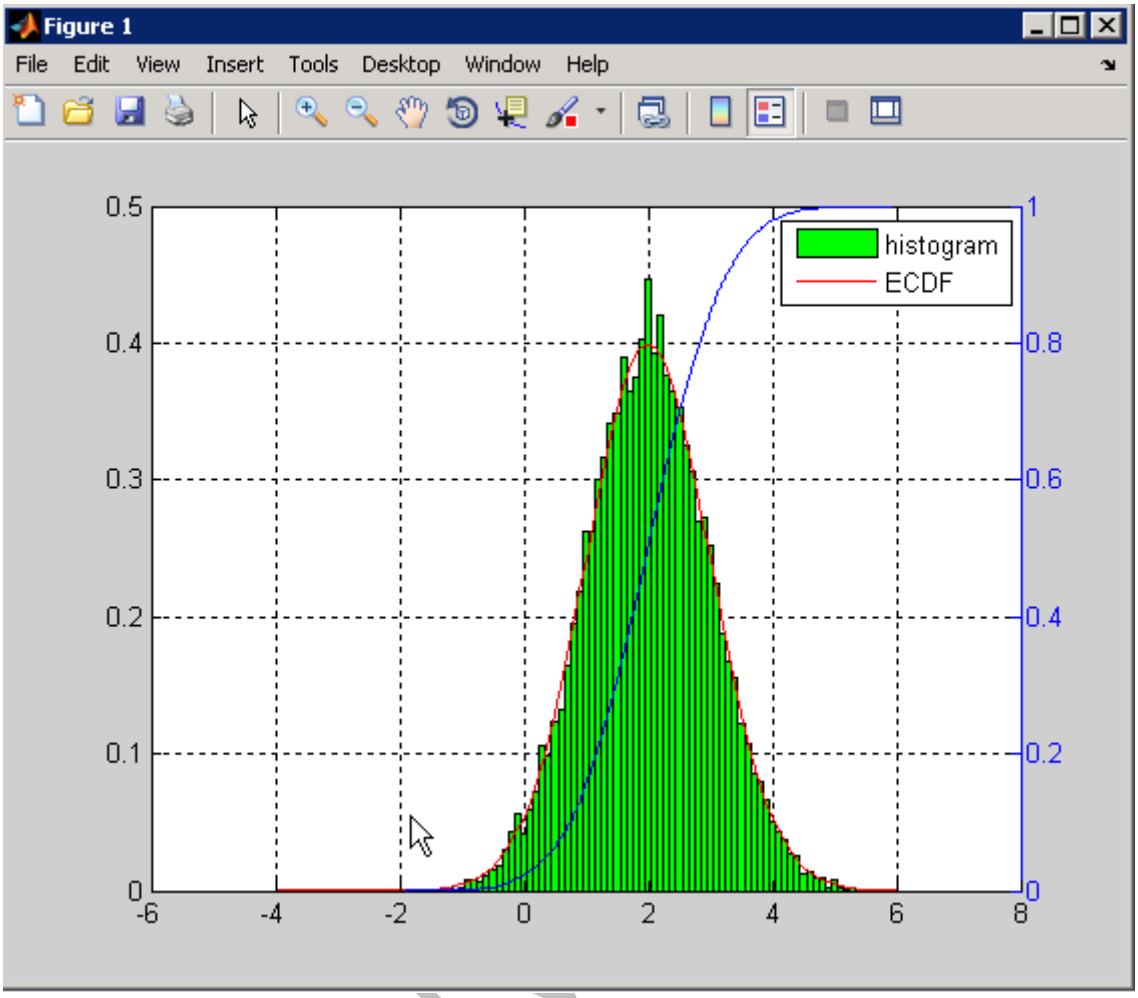
EDITOR PUBLISH VIEW

```
%Reminder: For a probability density function, the integral over the entire space is 1.
%Dividing by the sum will not give you the correct density.
% To get the right density, you must divide by the area.

nb_Simulation=10000;
mu=2;sigma=1;
y=normrnd(mu,sigma,nb_Simulation,1);
x=-4:0.1:6;
n=hist(y,x);

[y2,x2] = ecdf(y);
[y1,x1] = hist(y,x);
plotyy(x1,y1,x2,y2,@(x,y)bar(x,y/trapz(x,y),1,'g'),'stairs');
hold on;
plot(x,normpdf(x,mu,sigma),'r');
legend('histogram','ECDF');
grid on;
```

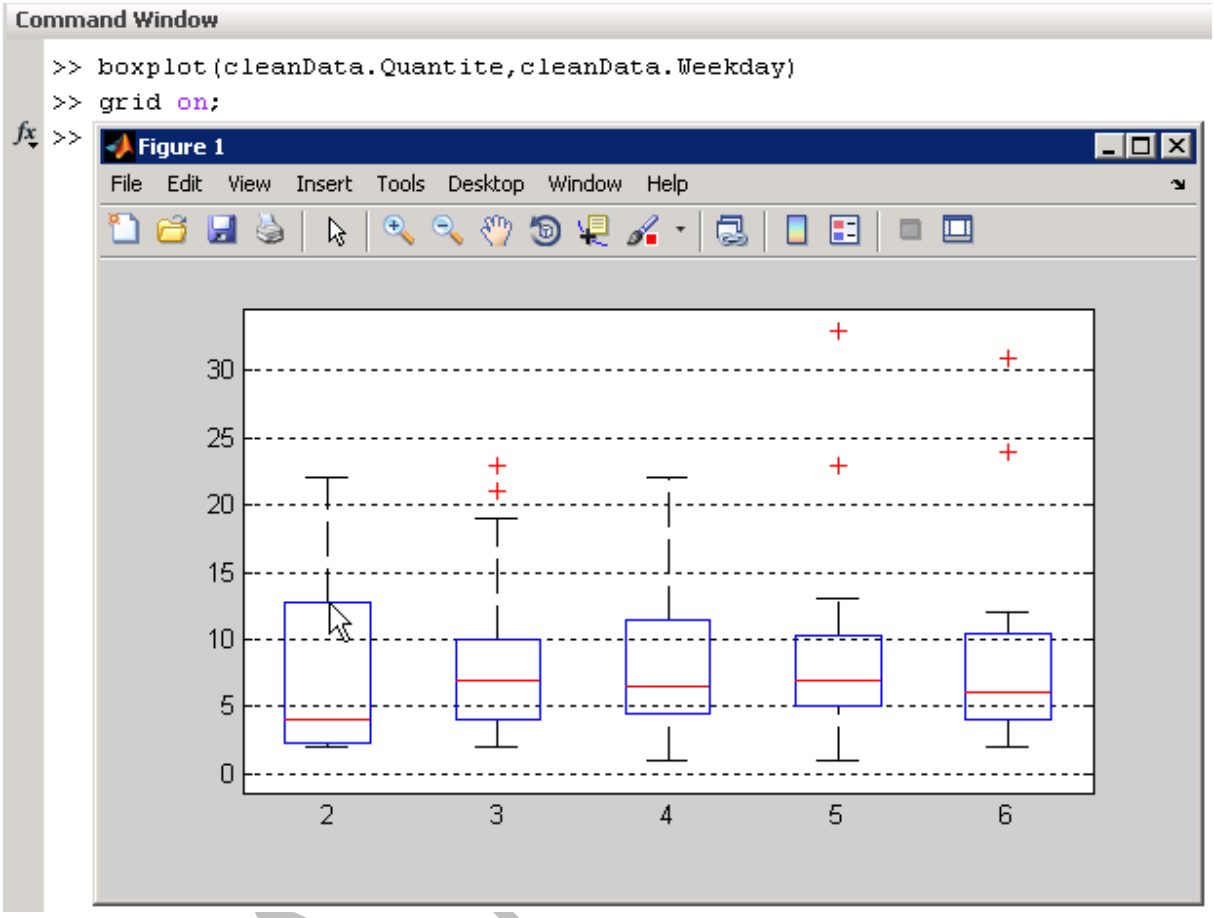
The result will be:



SAM

Box-Whisker plot

Compare to R, the Box-Whisker plot in MATLAB are very poor in possibilities and fine tuning but let us see an example using the function **boxplot( )**:



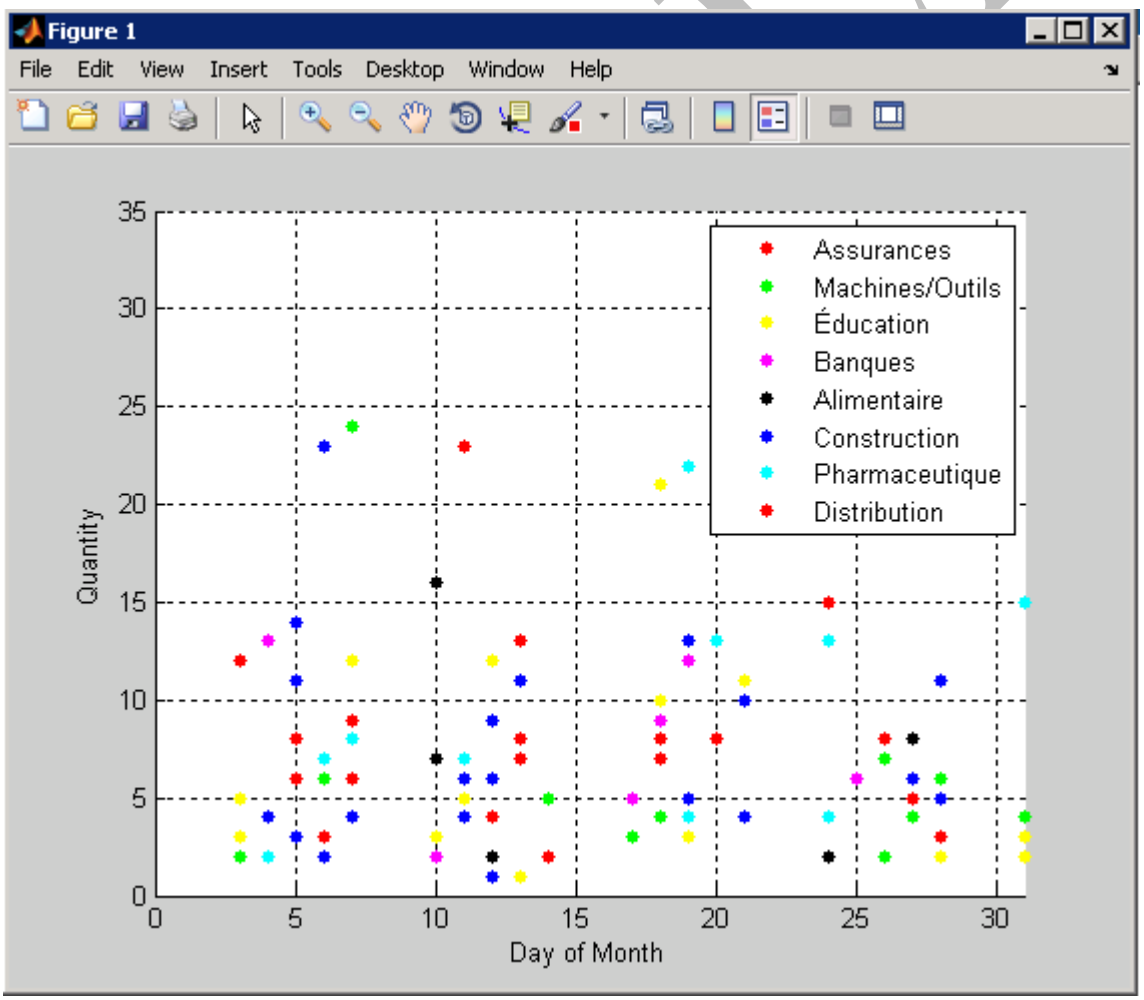
SA

Scatter Group plot

To simplify the management of multiple dot plots with datasets, MATLAB Statistical Toolbox has a **gscatter( )** function to simplify a little bit the fact that we often need to plot groups of points with different colors:

```
1    gscatter(cleanData.Day,cleanData.Quantite,cleanData.Activite,'rgymkbc')
2    %there is not enough color for the last group of Activity...
3    %this is why gscatter sucks
4    xlabel('Day of Month');ylabel('Quantity');
5    xlim([0 31]);
6    grid on;
```

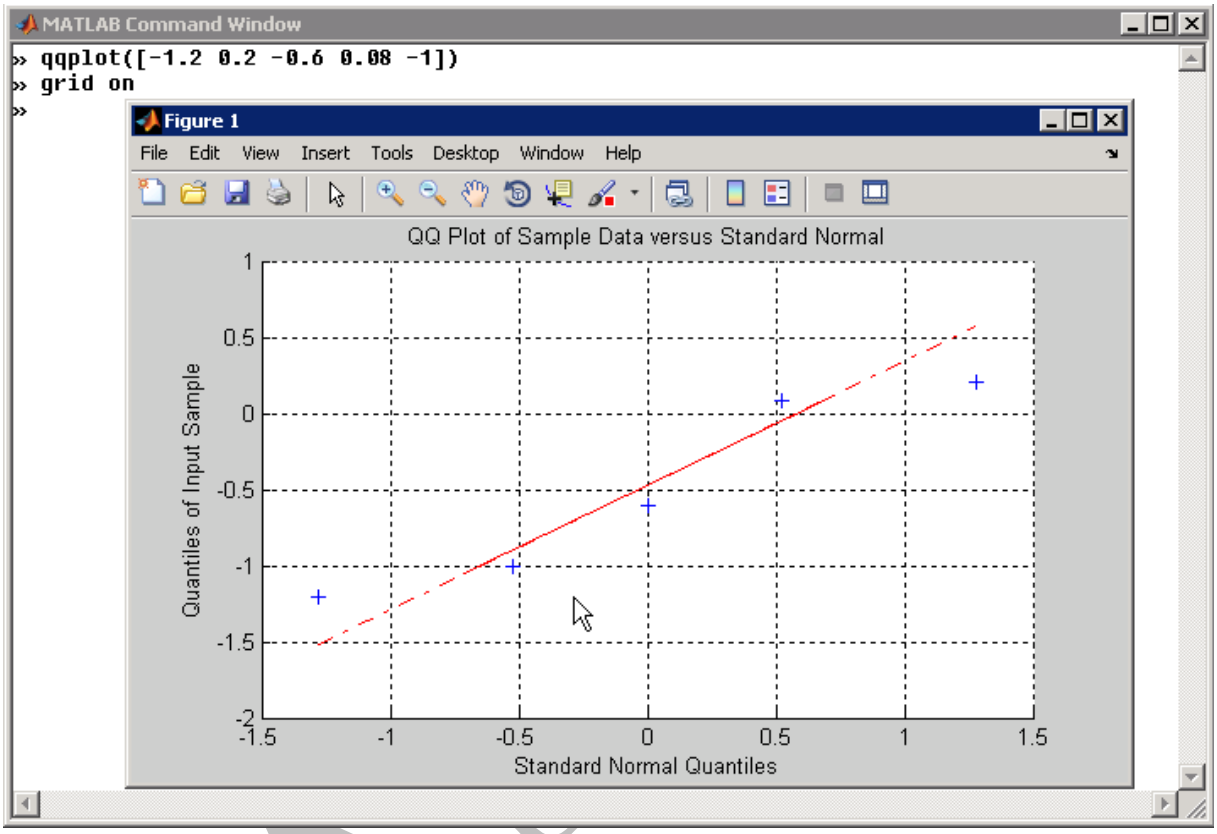
The result will be:



QQ plot

QQ plot are often used to check qualitatively the normality of data. For this purpose we have a **qqplot( )** function in the MATLAB Statistical Toolbox:

The main purpose of this example is just to compare the result with those of Minitab and R using the same data:

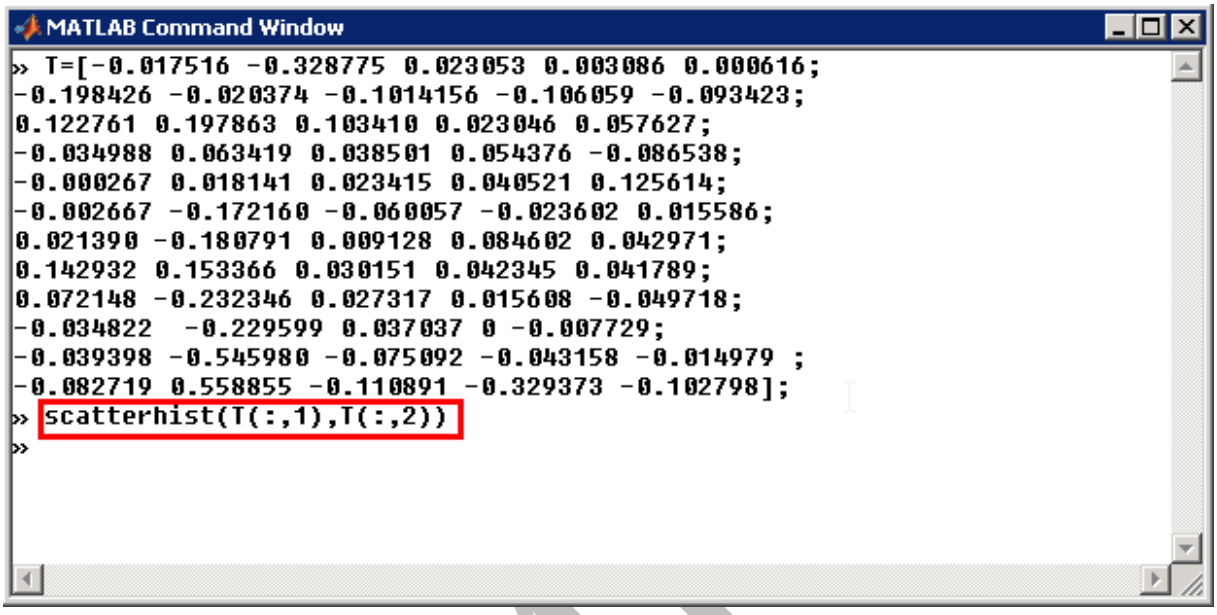


SAMPLE

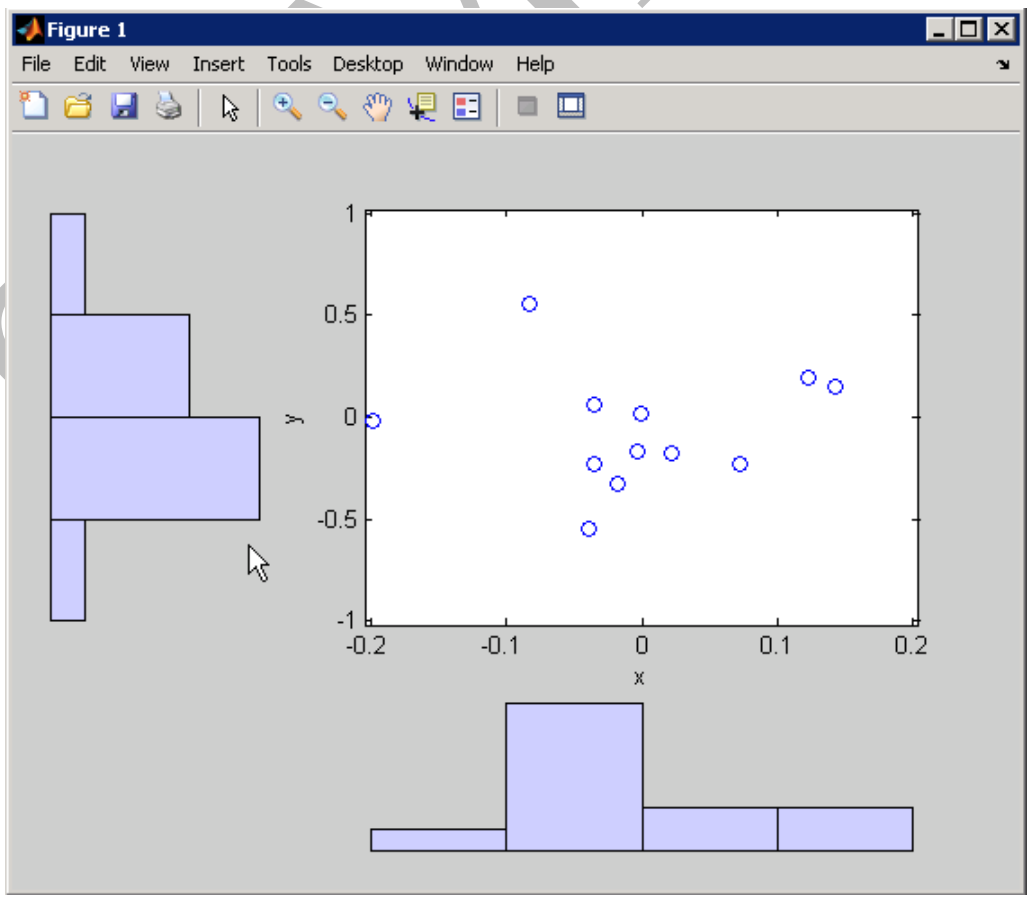
Histogram Scatter

Histogram scatters are very useful plots in finance and very flexible in R. But has such a plot with the function **scatterhist( )** but the result is very hard to customize in comparison of R.

Using our asset data sample:



We get:



Copulas

Copulas is a major subject if finance and because in the theoretical course we made only the proof of the Gaussian copula and its properties we will focus only on this one...

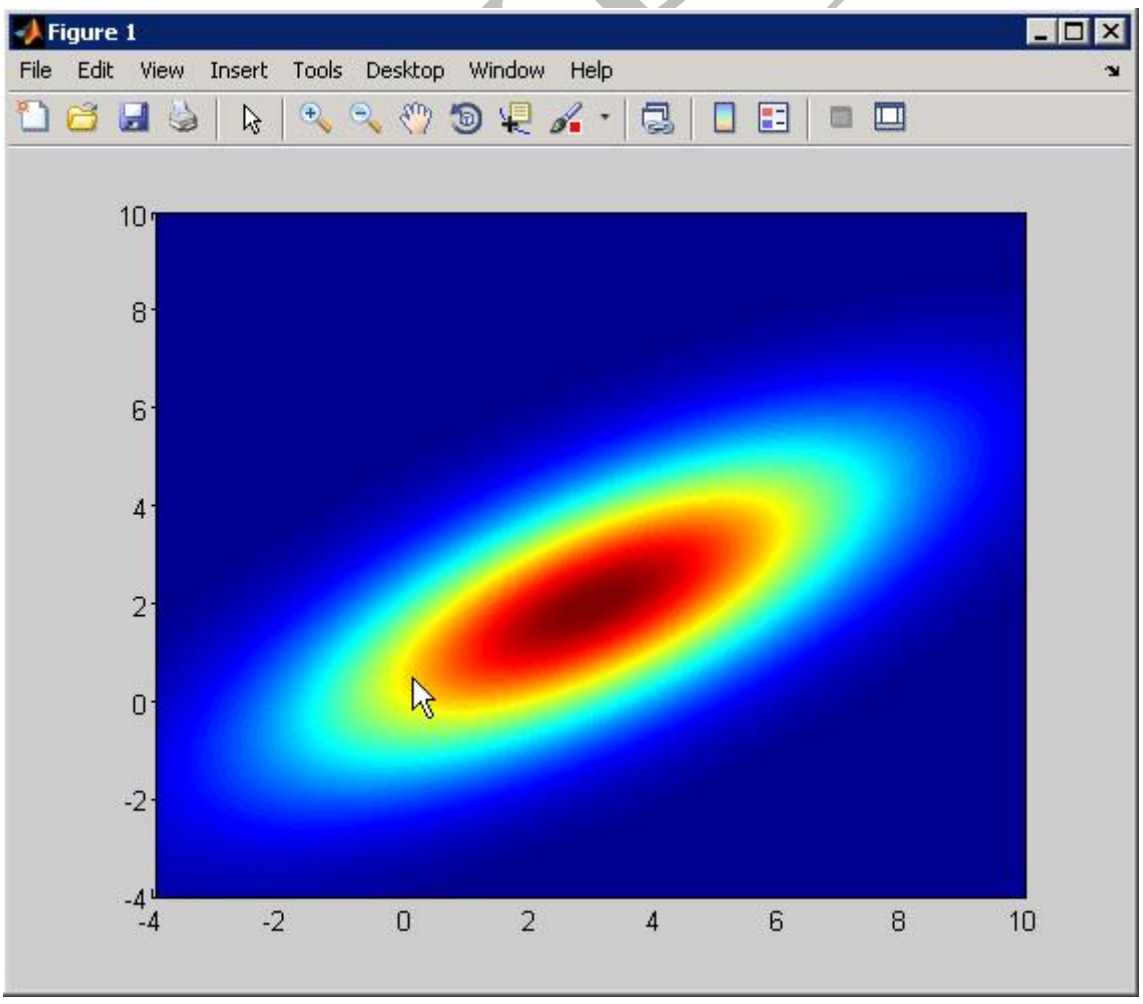
Gaussian copula

First, to simply simulate the same Gaussian copula as in the theoretical we will use the functions **mvnpdf( )** of the Statistical Toolbox and **pcolor( )**:

EDITOR PUBLISH VIEW

```
1    [X,Y]=meshgrid(-4:.05:10,-4:.05:10);
2    M=[3,2];
3    V1=[10 5;5 5];
4    Z1=mvnpdf([X(:),Y(:)],M,V1);
5    pcolor(X,Y,reshape(Z1,length(X),length(X)));
6    shading interp;
```

The result will be (exactly the same as the one obtained with R and Maple):

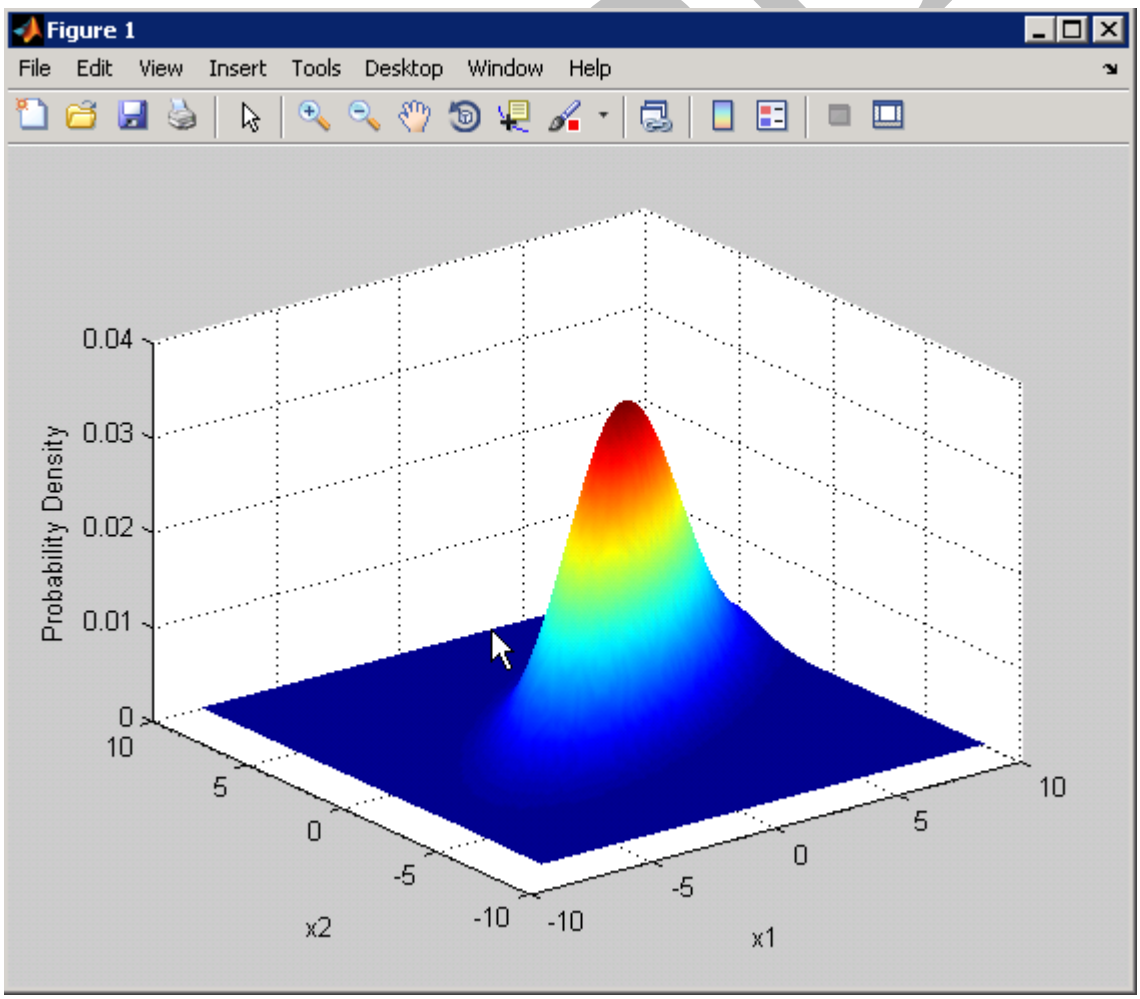


And what about the 3D view?

This is a hard to guess script using the function **reshape( )** in addition to the previous script:

```
1     mu = [3 2];
2     Sigma = [10 5;5 5];
3     x1 = -8:.2:10;
4     x2 = -8:.2:10;
5     [X1,X2] = meshgrid(x1,x2);
6     F = mvnpdf([X1(:) X2(:)],mu,Sigma);
7     F = reshape(F,length(x2),length(x1));
8     surf(x1,x2,F);
9     xlabel('x1'); ylabel('x2'); zlabel('Probability Density');
10    shading interp;
```

We get the following result:



This is in adequation with what we get with R and Maple.

We can mix all the stuff below to get a global view using a subplot:

```
clc;clear;close all

mu=[3 2];
Sigma=[10 5;5 5];
[X,Y]=meshgrid(-10:0.1:10,-10:0.1:10);
p=mvnpdf([X(:) Y(:)],mu,Sigma);
p=reshape(p,size(X));

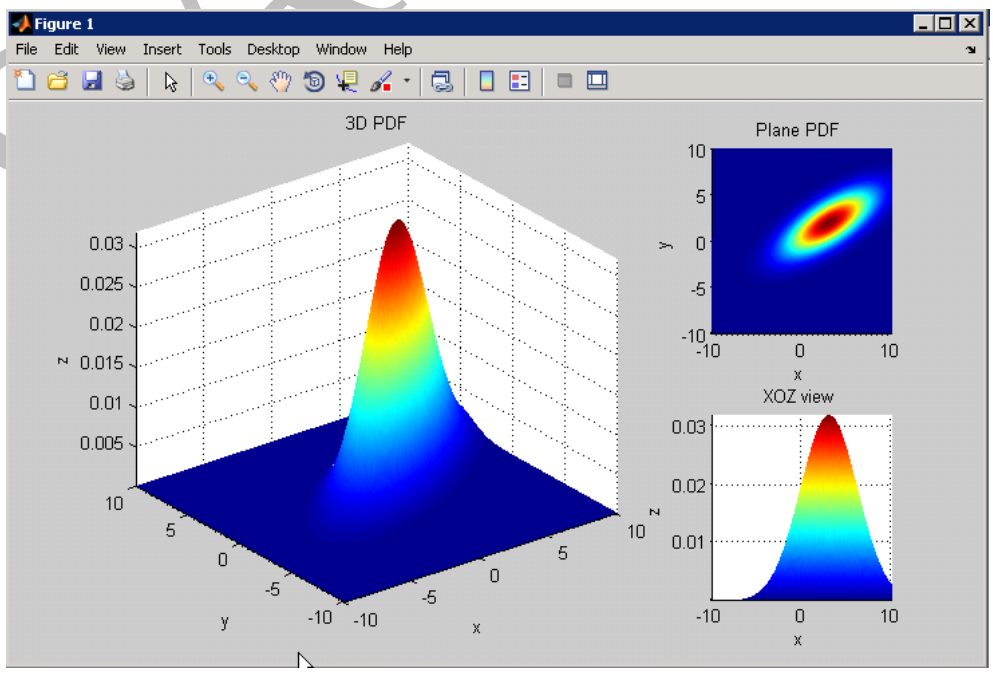
set(gcf,'Position',get(gcf,'Position').*[1 1 1.3 1])

subplot(2,3,[1 2 4 5])
surf(X,Y,p),axis tight,title('3D PDF')
xlabel('x');ylabel('y');zlabel('z');
shading interp;

subplot(2,3,3)
surf(X,Y,p),view(2),axis tight,title('Plane PDF')
xlabel('x');ylabel('y')
shading interp;

subplot(2,3,6)
surf(X,Y,p),view([0 0]),axis tight,title('XOZ view')
xlabel('x');zlabel('z')
shading interp;
```

The result will be:



Distributions

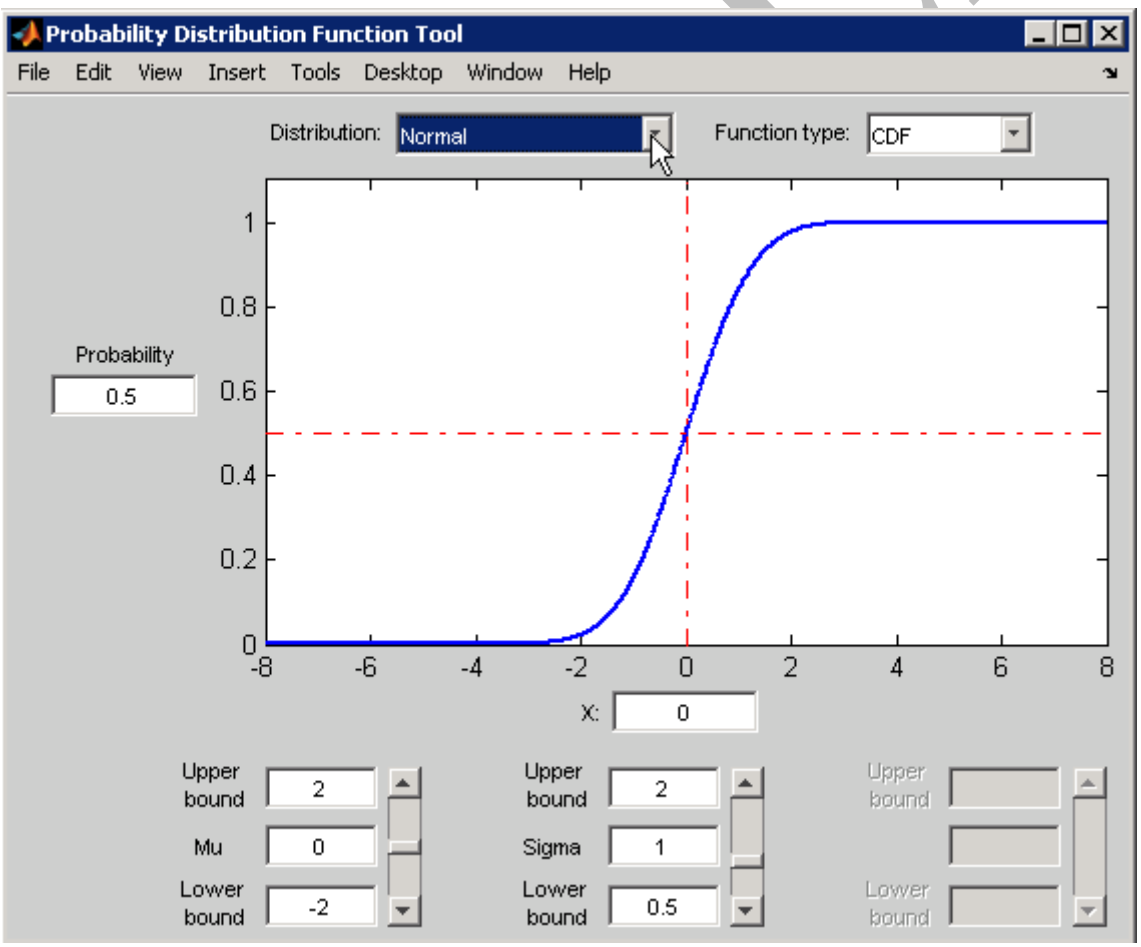
As we know it almost more than a hundred probabilities distributions are used in all domains of sciences (finance, pharma, econometrics, reliability, physics, etc.).

As we know that the MathWorks website describe well what is the syntax of each available distribution it is completely useless to redo examples for each of them.

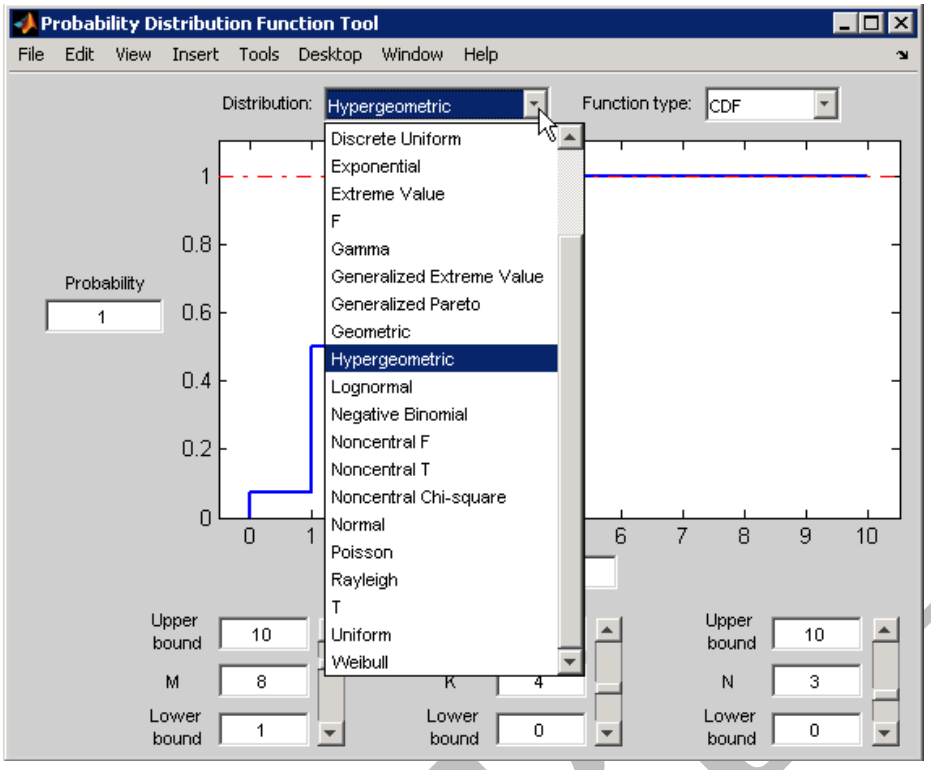
Thus we will focus only on one tool. The **disttool** that is more for academic purpose than anything else for demonstrating to students how looks like the shapes of various distributions. First open the tool:



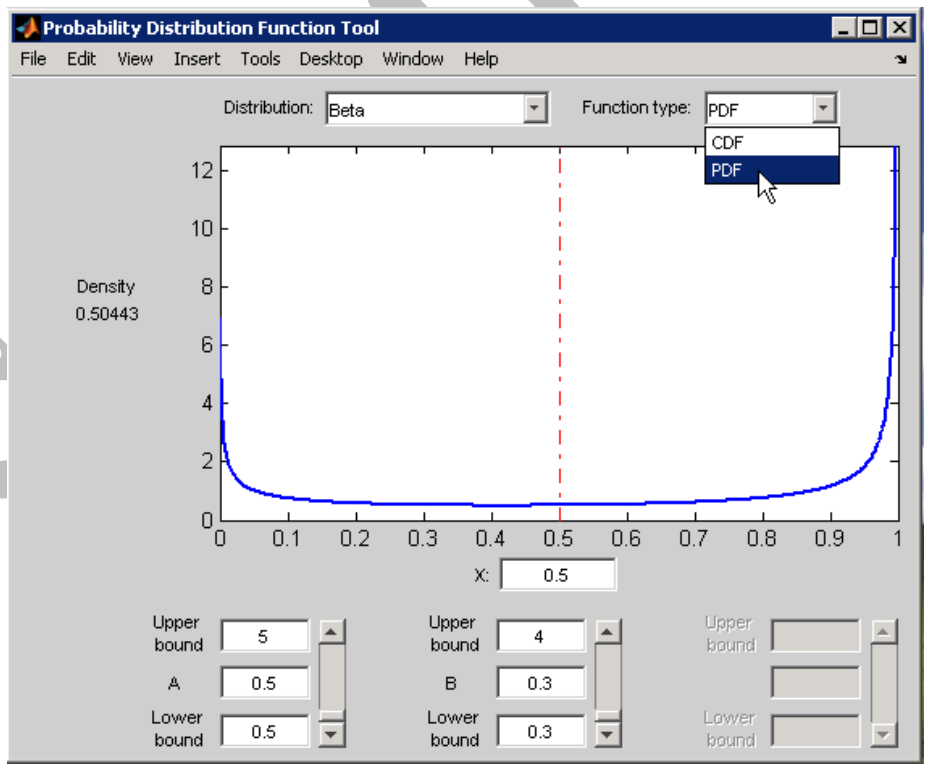
So appears:



We have some distribution choices:



And the possibility to choose between CDF and PDF:

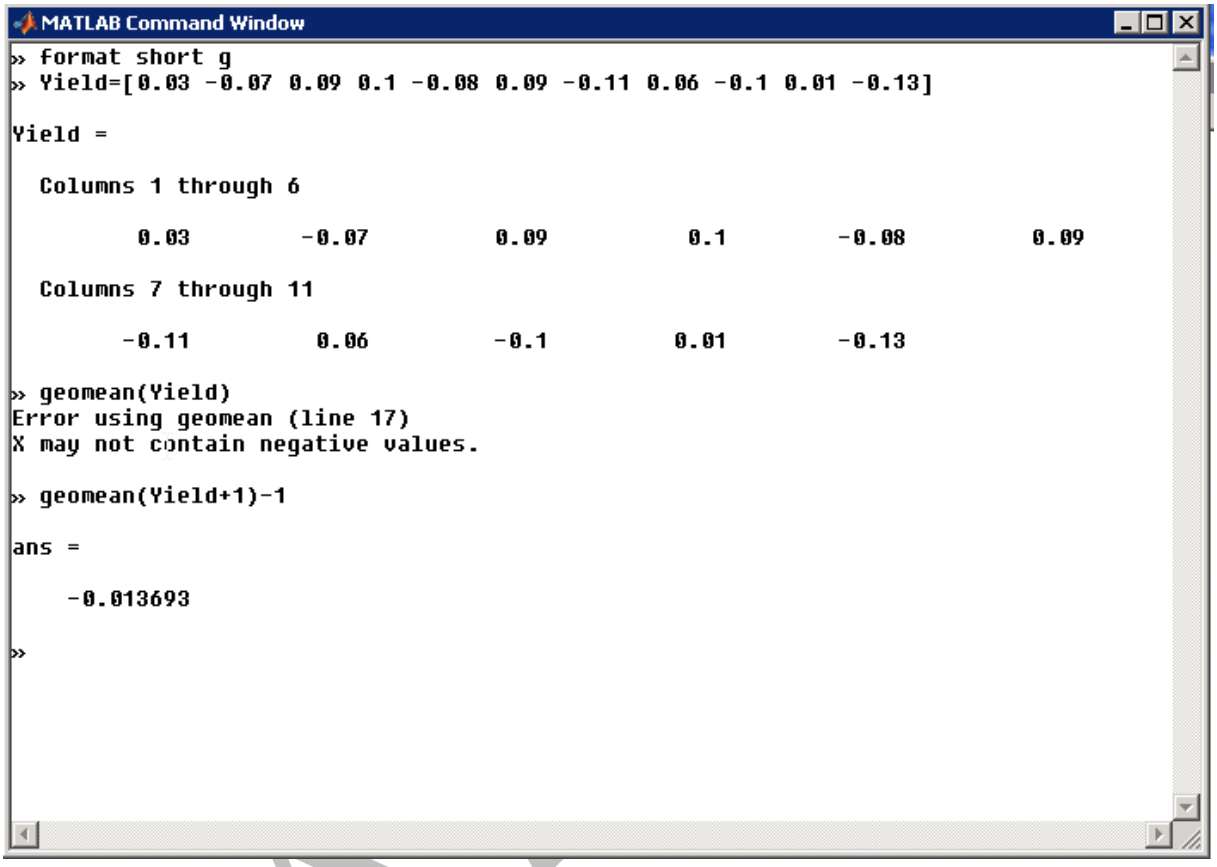


The **Upper bound** and **Lower bounds** fields visible are here to define each scrollbar limits if you want to play to change the distribution parameters with the scrollbars!

Descriptive statistics

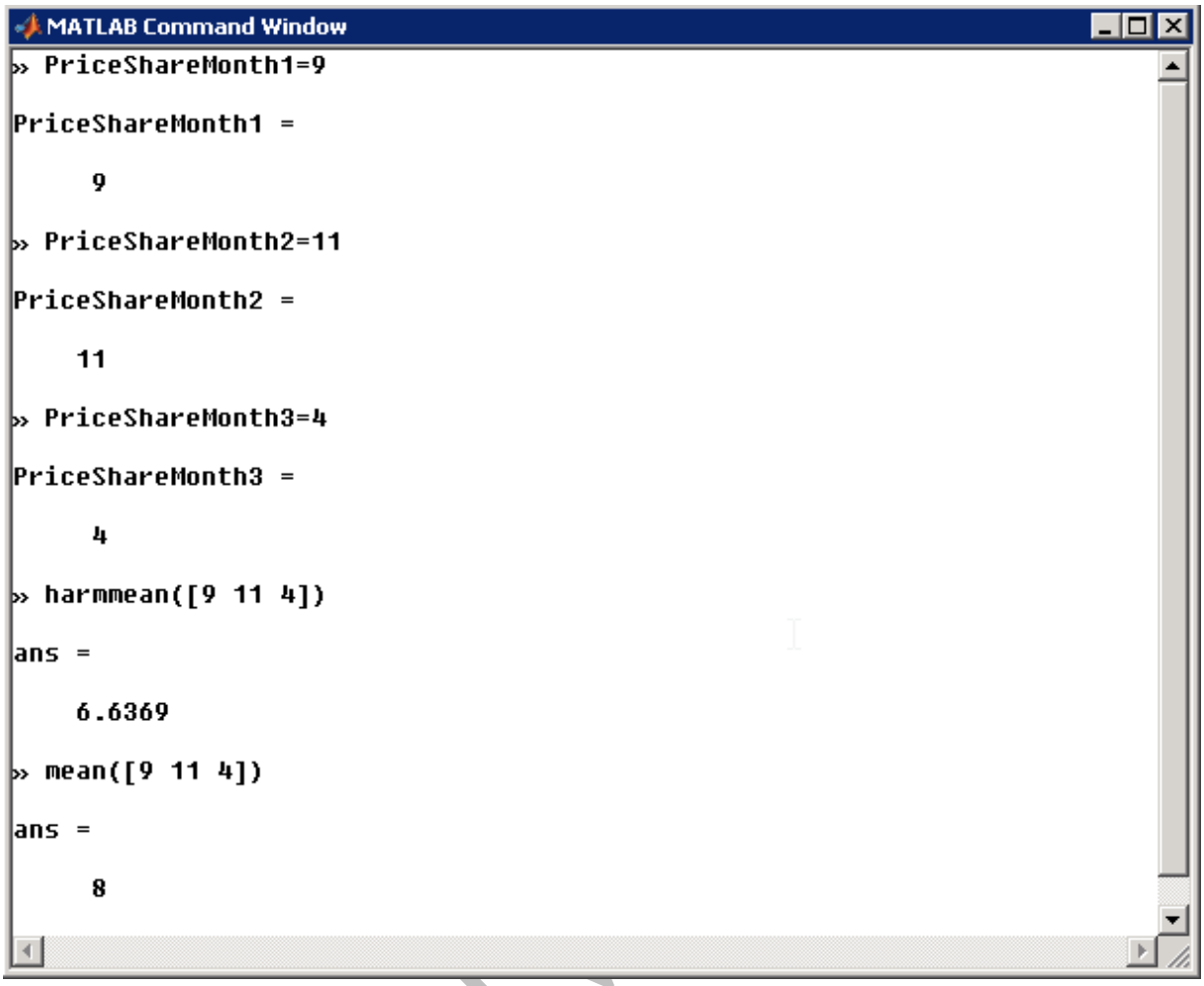
Now that we have seen the basic charts available in the Statistical Toolbox we will see some very elementary statistical calculations.

Geometric mean



SA

Harmonic mean

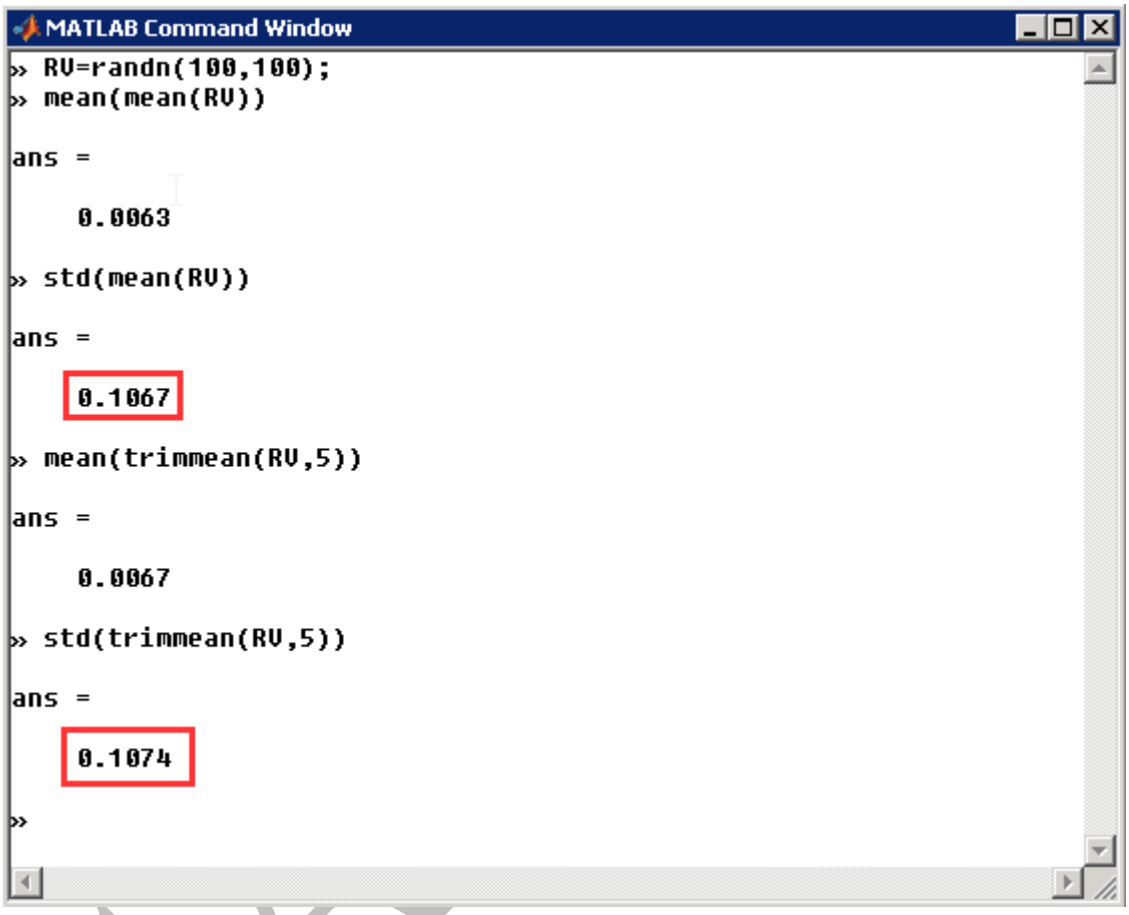


SAM

Trimmed mean (M-statistics)

In comparison of SPSS as far as I know MATLAB Statistical Toolbox is very poor in the integration of well know M-Statistics. The only integrated one seems to be the trimmed mean....

Then let us see how to use it with a trimmed of 5%:



CDF/iCDF/Quantile/Centile

Cumulated probability **cdf( )**, inverse cumulated probability **icdf( )** and also percentile **prctile( )**, **quantile( )** and **median( )** are use a lot in finance and in general in stochastic simulations for various purposes (Value At Risk, Threshold values of a sample, etc.).

Let us see a very academic example used to check the validity of the different aspect of those functions:

```
1   int_Simulations=1000;
2   mu=3;sigma=1;
3   Nx=normrnd(mu,sigma,1,int_Simulations); %N(3,1)
4
5   %Cumulated probability (has to be near form 0.5 in this example)
6   CDF=cdf('Normal',3,mu,sigma)
7   %inverse CDF
8   iCDF=icdf('Normal',0.5,mu,sigma)
9
10  %Value corresponding to 50% cumulated probability
11  prc=prctile(Nx,50)
12  quant=quantile(Nx,0.5)
13  med=median(Nx)% just a check :-P
14
15  %Cumulated probability of a simulation
16  x=3; % threshold
17  cProb=nnz(Nx(:)<=x)/int_Simulations
```

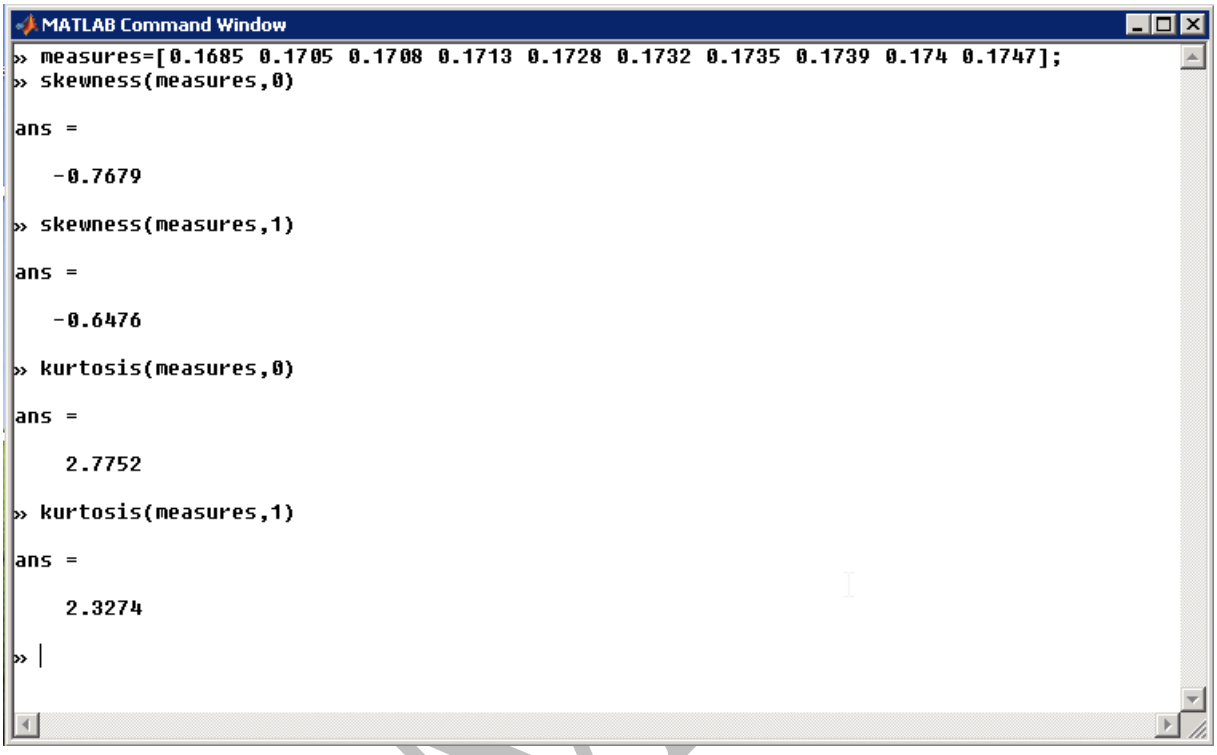
The result will give:

```
Command Window
CDF =
    0.5000
iCDF =
     3
prc =
    2.9876
quant =
    2.9876
med =
    2.9876
cProb =
    0.5030
fx >>
```

Skewness/Kurtosis

Since there are many methods to calculate the kurtosis (flatness) and the skewness[3] (asymmetry), I still want to treat them separately from the rest.

Consider the following examples:



In MATLAB, we have then the following rules::

- If the **skewness** is zero, the distribution is symmetric (this does not mean that the symmetry occurs on a peak of the distribution because in the case of a bimodal distribution the axis of symmetry can be balanced between the two modal values). If the skewness is positive, the distribution (modal/median) looked right (or there are extreme right values). If the skewness is negative, the distribution (modal/median) is leaning to the left (or there are extreme left values)

- If the **kurtosis** is equal to three (platikurtic) then flattening is similar to a Normal distribution. If the value is greater than 3, then the distribution of interest (leptokurtic) is then higher than that of a Normal distribution with an equal mean (conversely - mesokurtic - if kurtosis is obviously less than 3)

[3] At least 11 methods to calculate the Skewness. See the study of Tabor, J. (2010), "Investigating the Investigative Task: Testing for Skewness - An Investigation of Different Test Statistics and their Power to Detect Skewness," Journal of Statistics Education, 18, 1-13. www.amstat.org/publications/jse/v18n2/tabor.pdf

For information and for comparison given the different more common ways to calculate skewness and kurtosis (the first one is called "Fisher-Pearson skewness coefficient" and the second "adjusted Fisher-Pearson standardized moment coefficient"):

$$\gamma_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^3}{\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}\right)^3}$$

$$\gamma_2 = \gamma_1\sqrt{\frac{n}{(n-1)(n-2)}}$$

$$\gamma_3 = \gamma_1\left(\frac{n-1}{n}\right)^{3/2}$$

and:

$$\kappa_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^4}{\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}\right)^4}$$

$$\kappa_2 = \frac{(N-1)\left((N+1)\kappa_1 - 3(N+1)\right)}{(N-2)(N-3)} + 3$$

$$\kappa_3 = \frac{N(N+1)}{(N-1)(N-2)(N-3)}\kappa_1 - \frac{3(N-1)^2}{(N-2)(N-3)}$$

$$\kappa_4 = \kappa_1 - 3$$

$$\kappa_5 = \left(\kappa_4 + 3\right)\left(1 - \frac{1}{N}\right)^2 - 3$$

SAMPLE

We have by comparison with MATLAB, R, Mathematica, Minitab, IMB SPSS Statistics and Microsoft Excel:

Skewness	R MATLAB Mathematica Minitab, SPSS, Excel	-0.647(γ_1),-0.77(γ_2), -0.552(γ_3) -0.647(γ_1),-0.77(γ_2) -0.647(γ_1) -0.77 (γ_3)
Kurtosis	R MATLAB Mathematica Minitab, SPSS, Excel	-1.114(κ_5),-0.22 (κ_3),-0.67 (κ_4) 2.327 (κ_1),2.775(κ_2) 2.327(κ_1) -0.22 (κ_3)

SAMPLE

Hypothesis tests

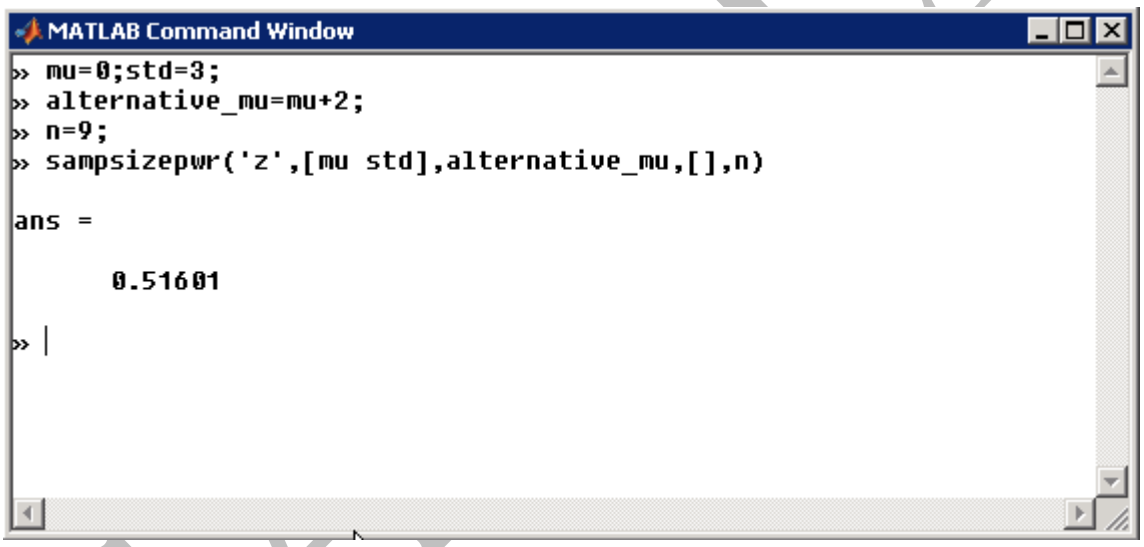
In this chapter we will also use the same data as in the Microsoft Excel, Minitab, SPSS and R training to be able to compare the results with the different e-books and videos tutorials. The purpose is also to compare the results with those obtained by hand during the proof of the different mathematical models!

Tests power, sample size and resolution

We have seen (proved) in the theoretic course the importance of the calculation of the a priori or a posteriori calculation of the power of some statistical tests and use the mathematical results to also be able to determine the necessary sample size or the resolution of a test.

Power of a Z test

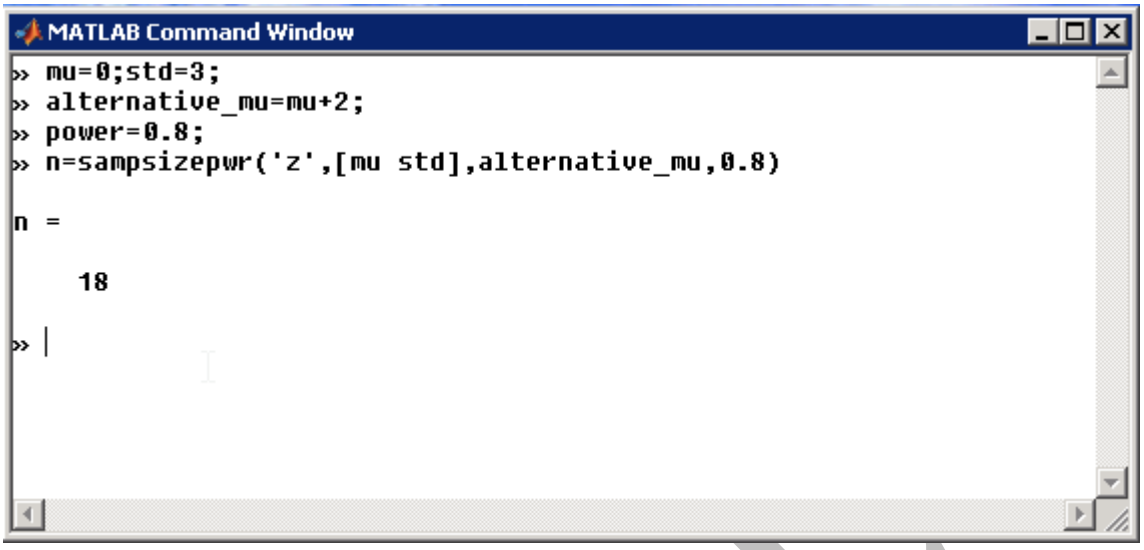
First have a look if we found with MATLAB the same power for a Z-test as the one calculated by hand or with Minitab. For this purpose we use the function **sampsizepwr( )**:



We get exactly the same result! That's a good thing!

Sample size of a one sample Z test

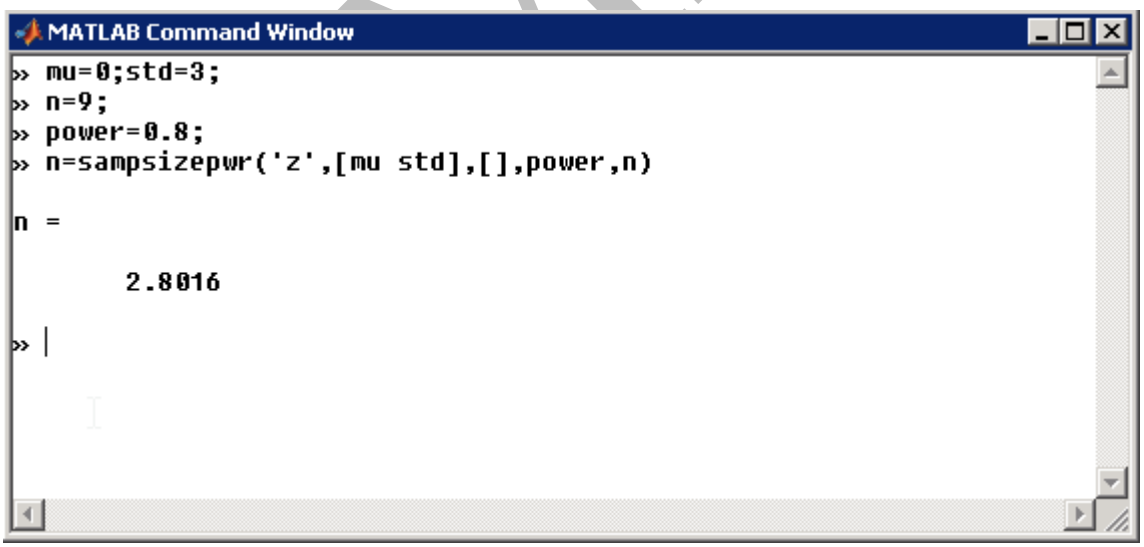
And what about the sample size? For this purpose we also use the function **sampsizepwr( )**:



We also get the same result!

Difference (resolution) of a one sample Z test

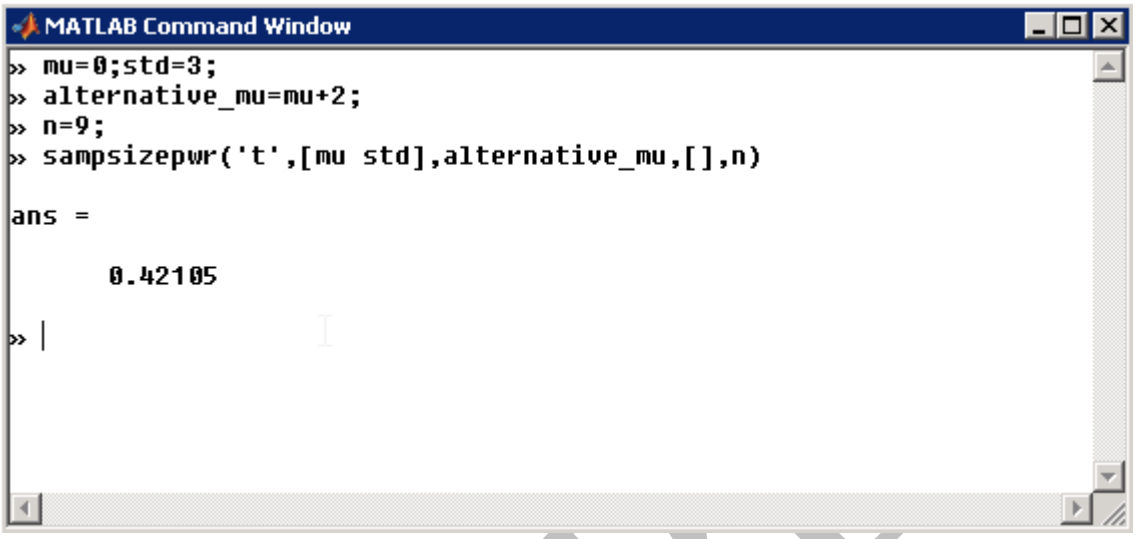
And what about the difference (resolution) ot the test? For this purpose we also use the function **sampsizepwr( )**:



We also get the same result!

Power of a t test

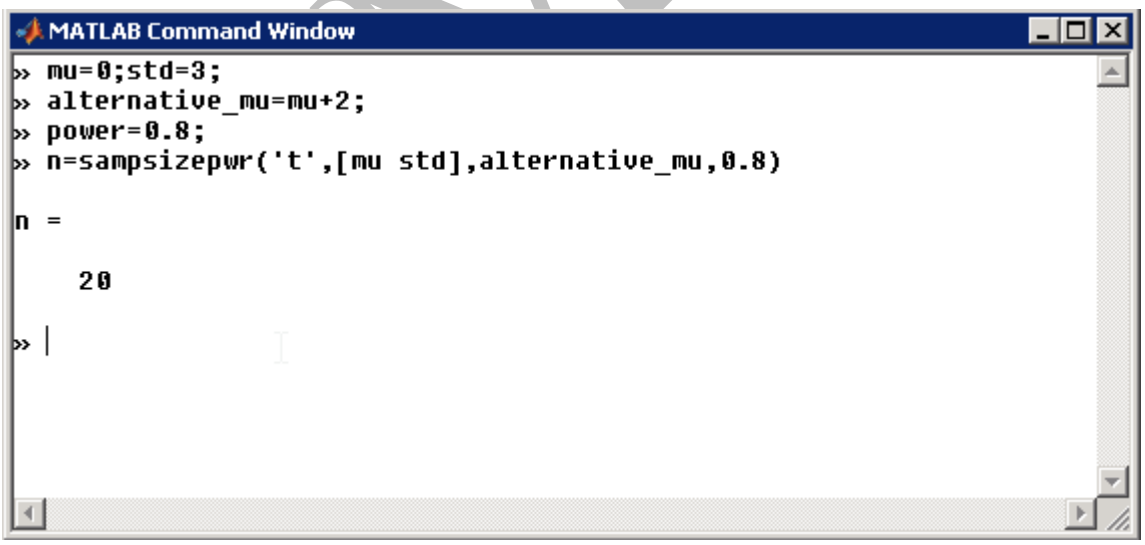
In the theoretical statistic course we also made the proofs for the t-test. Then have a look if we found the same results as the one calculated by hand and in Minitab always by using the function **sampsizepwr(** :



We also get the same result!

Sample size of a one sample t test

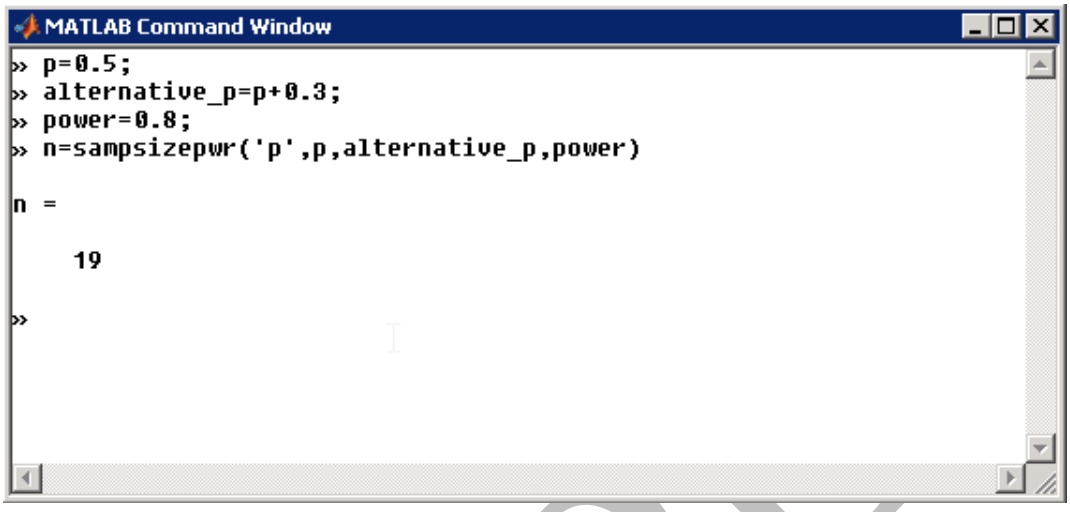
And what about the sample size? For this purpose we also use the function **sampsizepwr( )**:



We also get the same result!

Sample size of a one sample p test

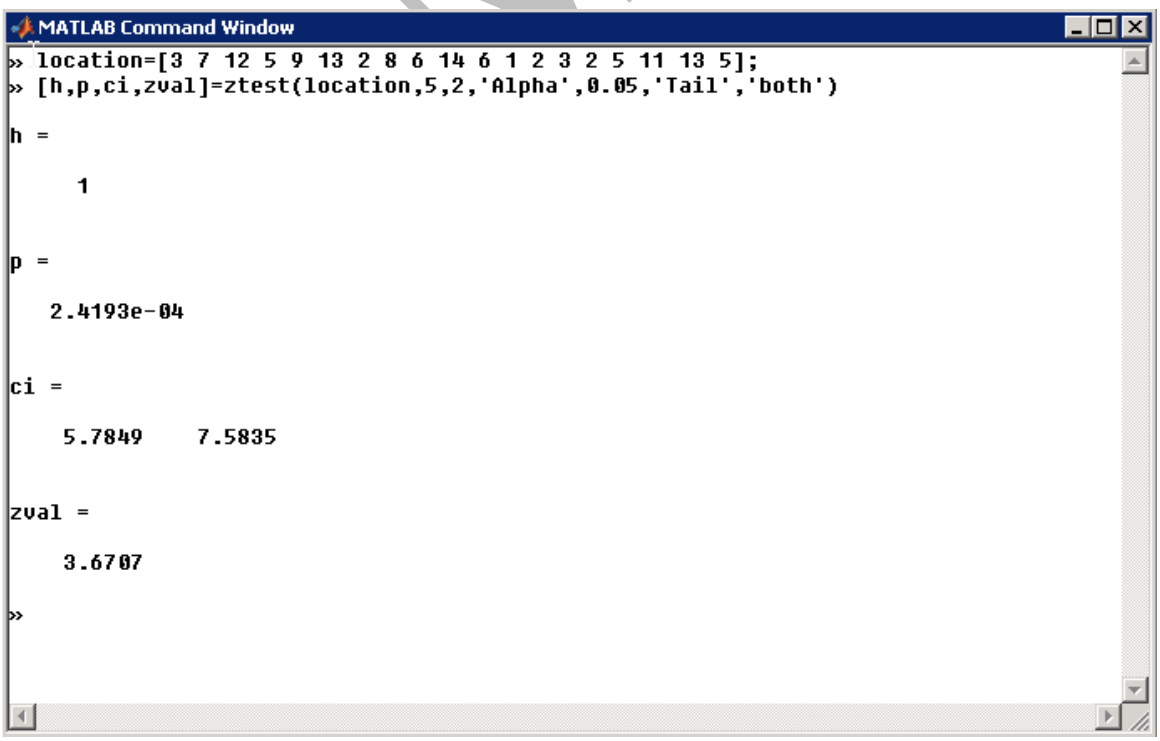
In the theoretical statistic course we also made the proofs for the p-test but only for the sample size. Then have a look if we found the same results as the one calculated by hand and in Minitab always by using the function **sampsizepwr(** :



That's right!

One sample Z-test

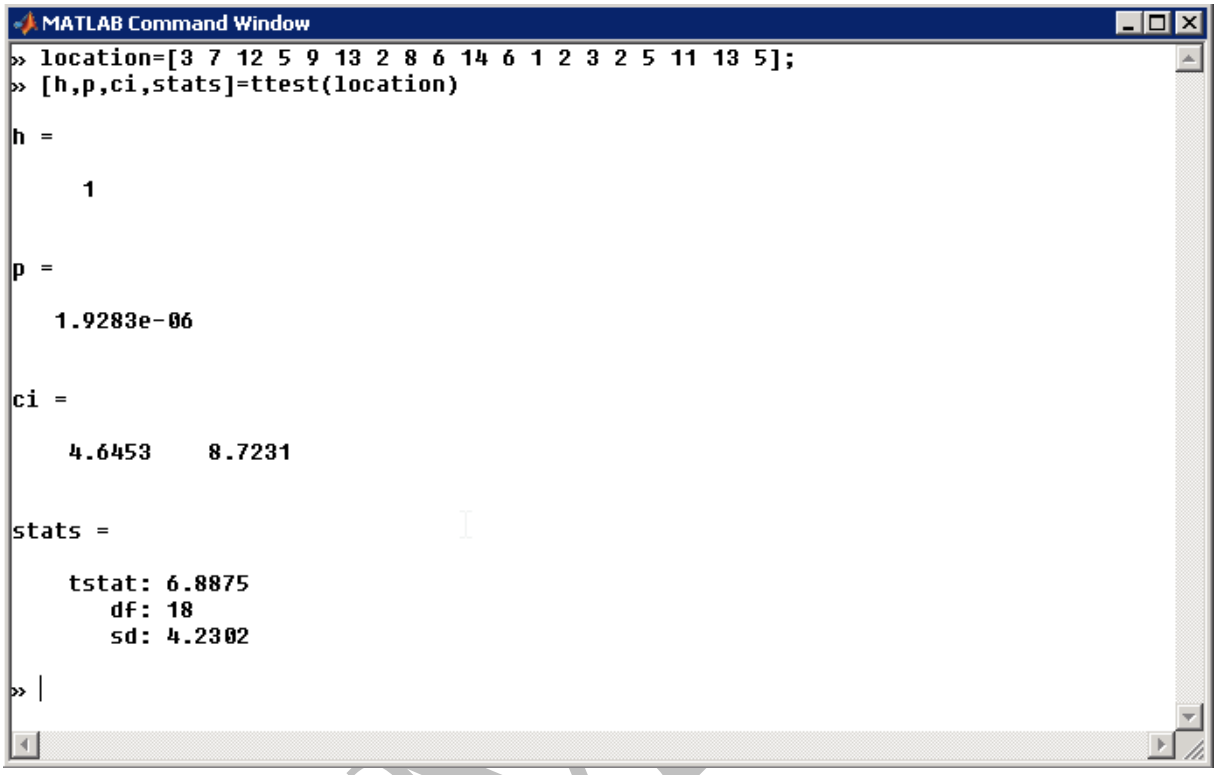
Using the function **ztest( )** let us see if we found the same p-value, confidence interval and z value as in Minitab, Excel and the calculations made by hand after the theoretical proof in the statistic course:



Yep! Everything is OK!

One sample t-test

Using the function **ttest( )** let us see if we found the same p-value, confidence interval and z value as in Minitab, Excel and the calculations made by hand after the theoretical proof in the statistic course:

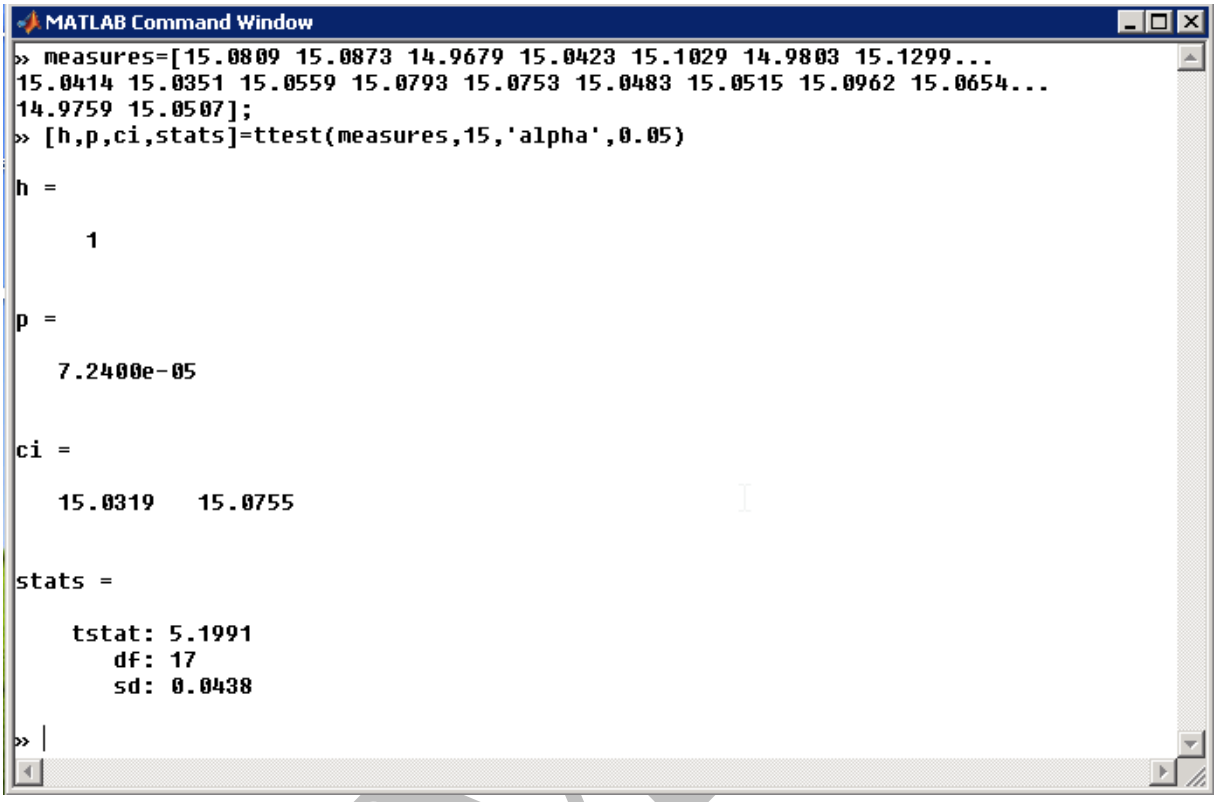


Yep! Everything is OK!

SAM

One sample t-test with explicit target mean

And what about the one sample t-test with explicit target mean? Let us check if we found the same result as the one calculated by hand and with Minitab and R?

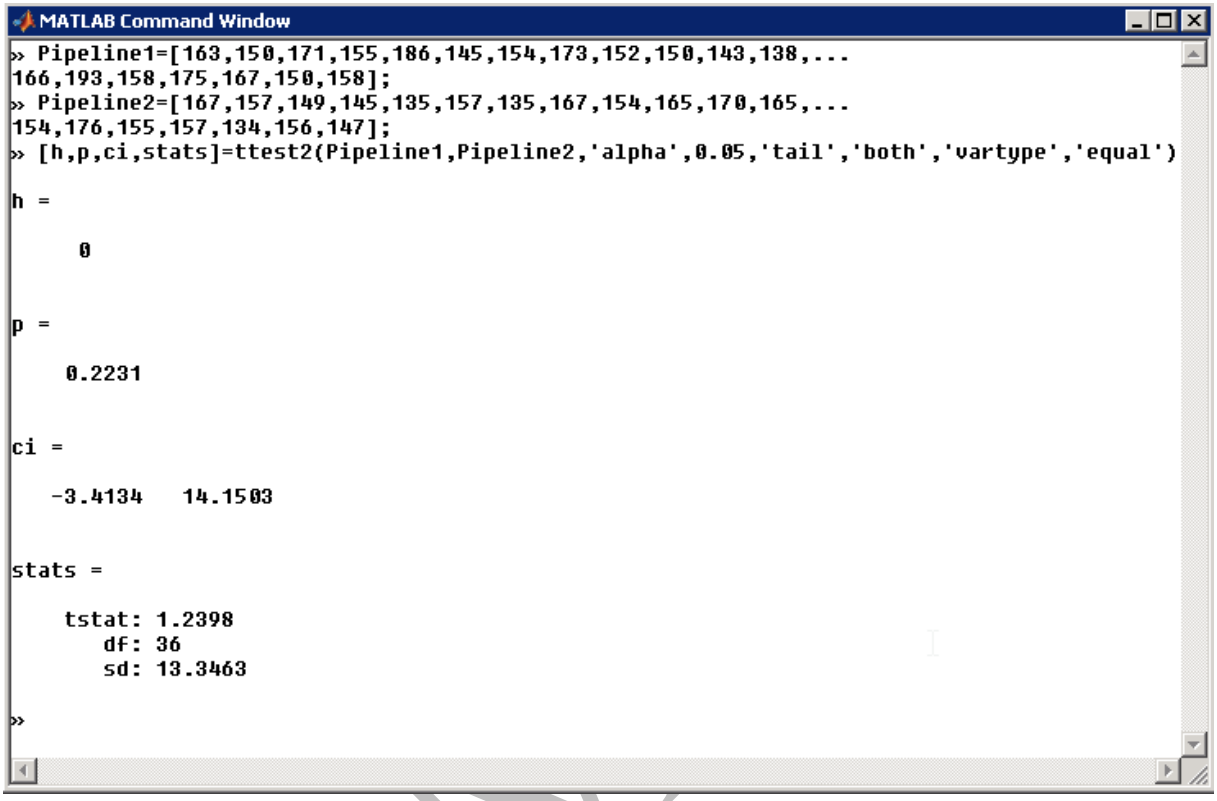


Yes we have the same result!

SAM

Two sample homoscedastic t-test

We have proved during the theoretic statistical course how to compare the mean of two unpaired samples under the hypothesis of homoscedasticity. The purpose now is to check if we obtain the same result as by hand on Minitab and R using the function **testt2( )**:

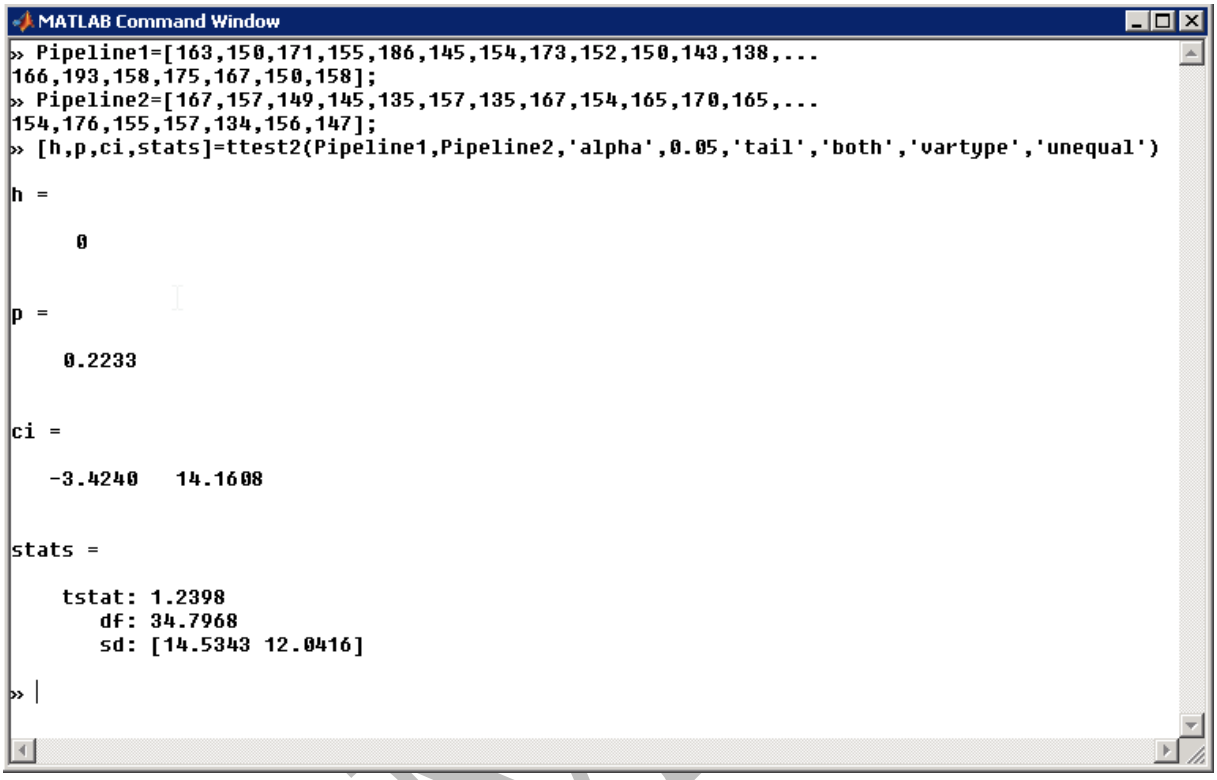


...and yes! Everyhting correspond!

SAM

Two sample heteroscedastic t-test (Welch test)

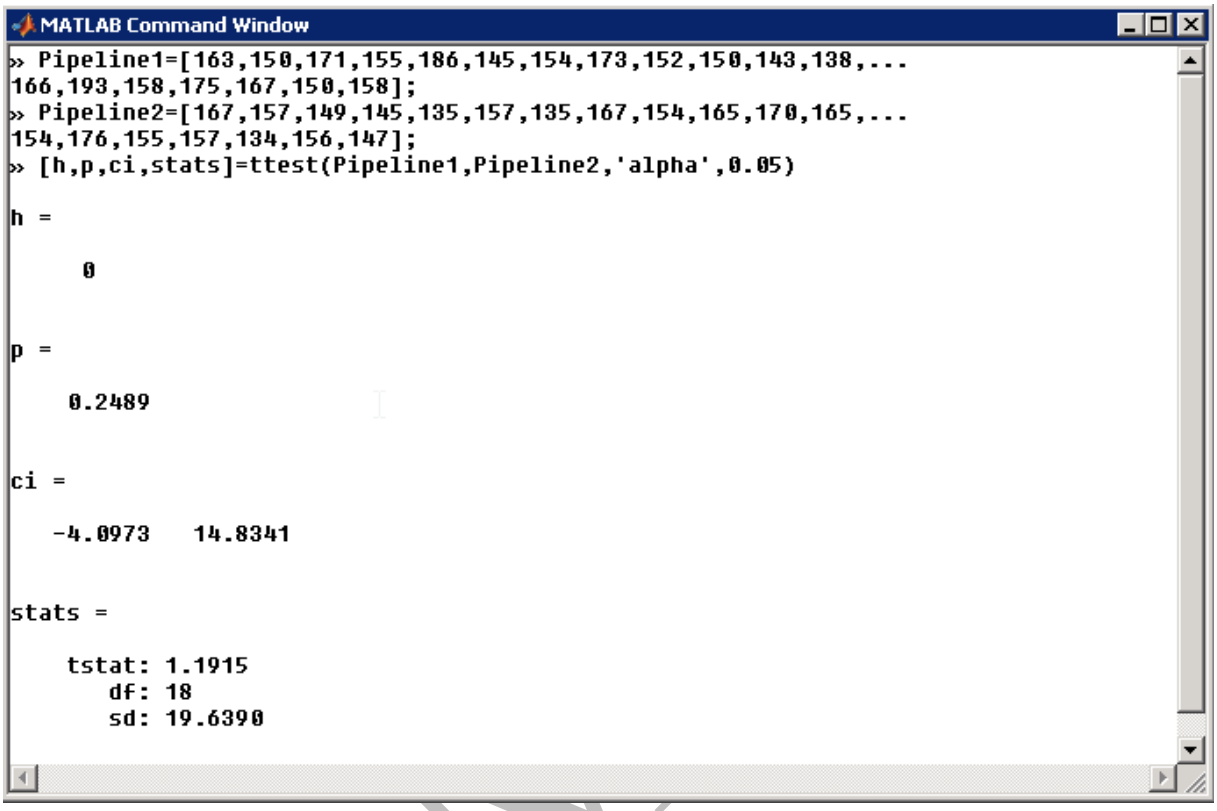
He have also proved in the theoretical course how to manage the case if the two samples don't have the same standard deviation. Then the purpose here is to check of we get the same results as those obtained by hand or with Minitab and R:



SAM

Paired t-test

We have also proved in the theoretical course how to compare the mean of two samples if the samples are paired. We will see now if we obtain the same calculations as by hand and Minitab and R:



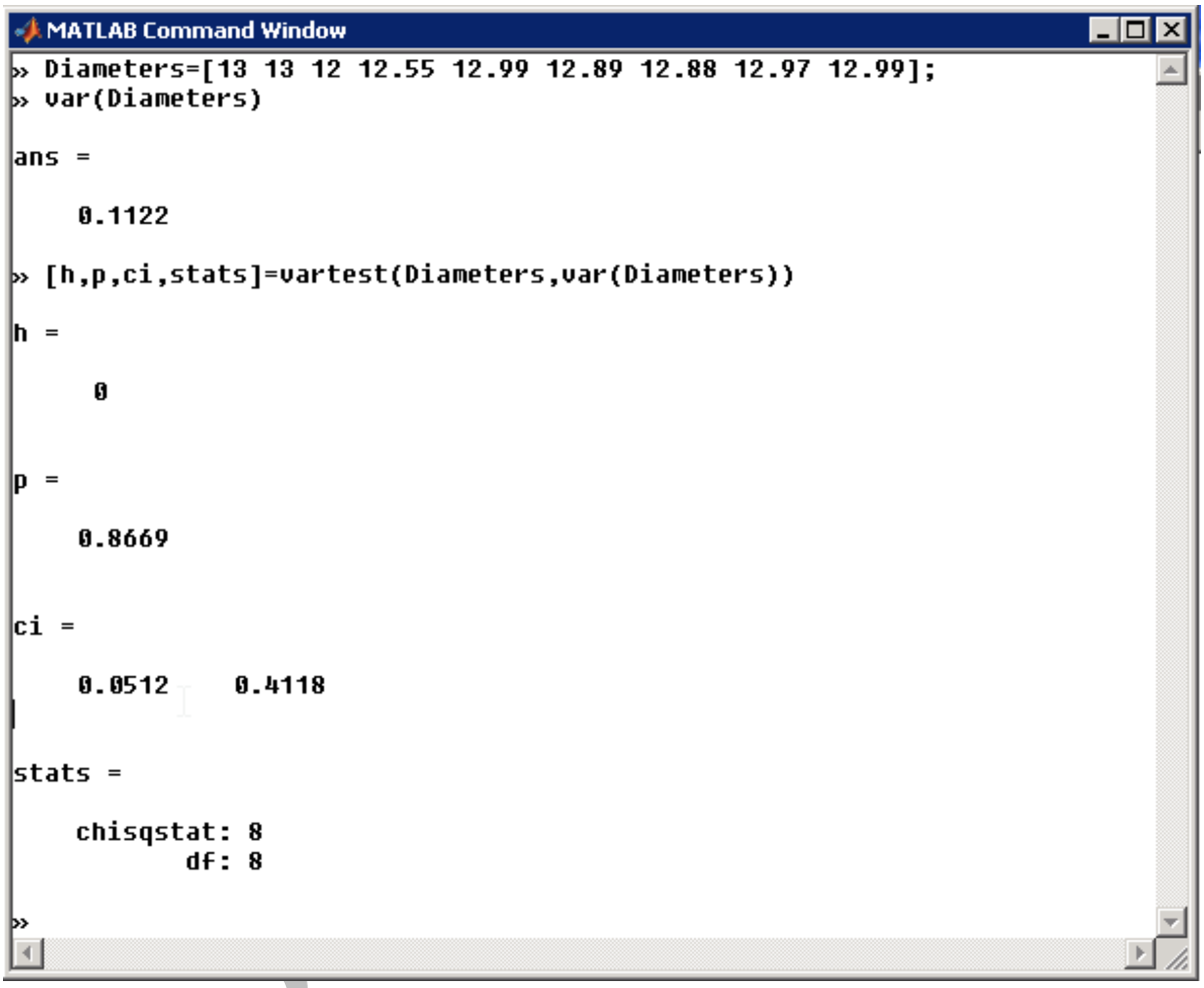
And the answer is: Yes! We obtain the same results.

SAM

Chi-square variance test

When we finished studying elementary tests for the comparisons of mean in the theoretical statistic course we started (and finished...) proofs on the comparison and interval of confidence of the variance. The first test we saw was the chi-square variance test and can be calculated in MATLAB with the function **vartest( )**.

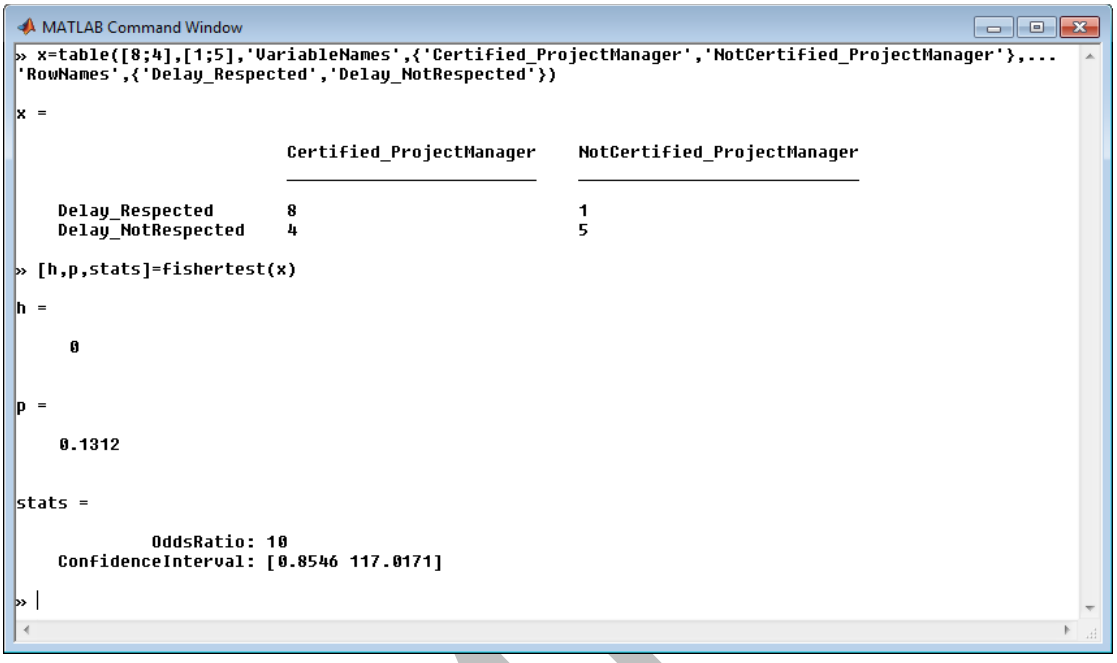
We take the same values as in theoretical course (and R+Minitab course) to check if we obtain the same results:



And that's it!

Exact Fisher Test

This is a new test of the version MATLAB 2015a. So let us do the same example as in the theoretical course (and also Minitab and R) using the function **fishertest( )**:



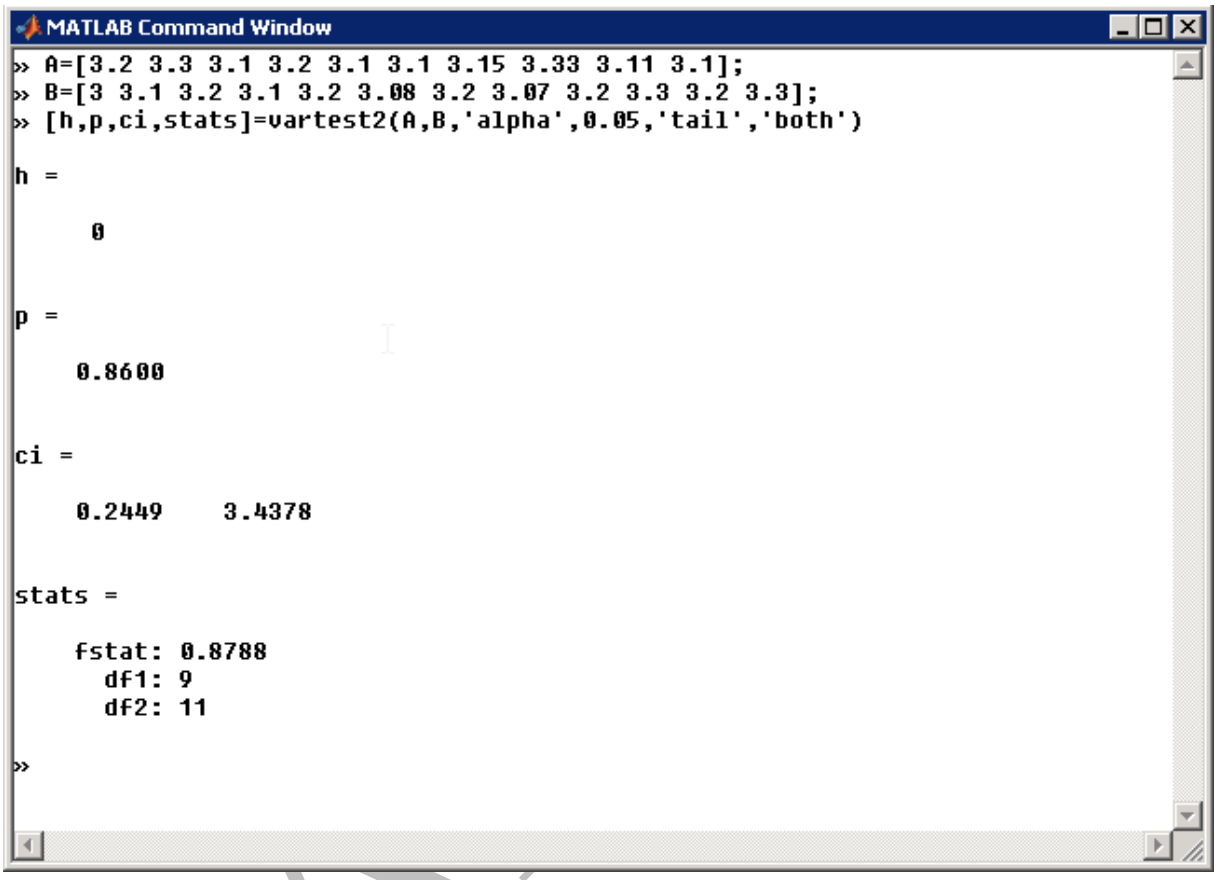
The *p*-value is correct but the odds ratio and confidence interval far away of what we get by hand and with R.

SAMPLE

Two sample F-test for equality of variances

And what about the ratio of two variances named "Fisher test" as we proved it in the theoretical statistic course?

Taking the same values as for calculated by hand and Minitab and R, we get in MATLAB using the function **vartest2( )**:

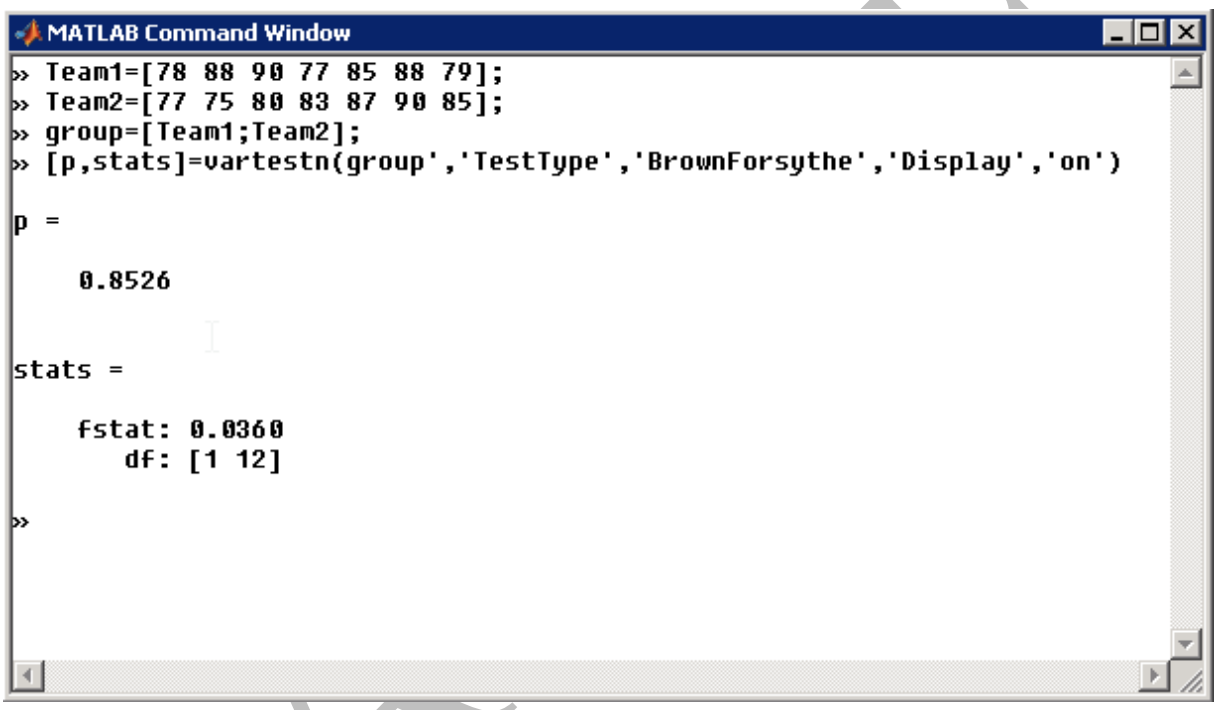


Two sample Levene's test for equality of variances

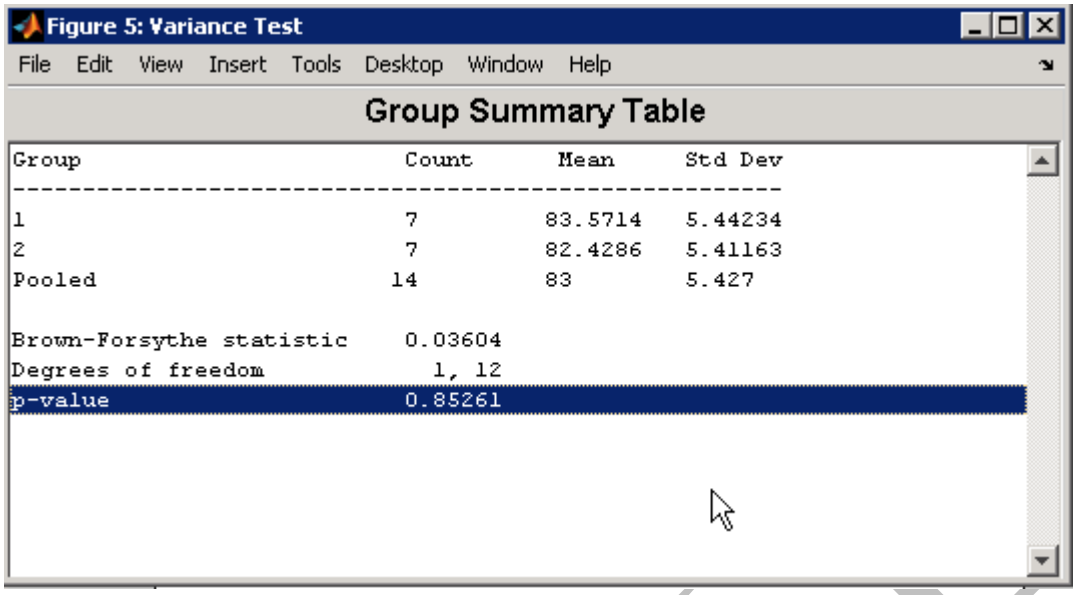
We have discussed in the theoretical statistic course about some problems of robustness of the test of the comparison of variance. Different empirical model of robustness were proposed to solving this issue (almost fifty tests classified under the name Test for Homogeneity of the Variance or "THV"). We will see here some of the most famous one available in MATLAB.

Brown-Forsythe (median)

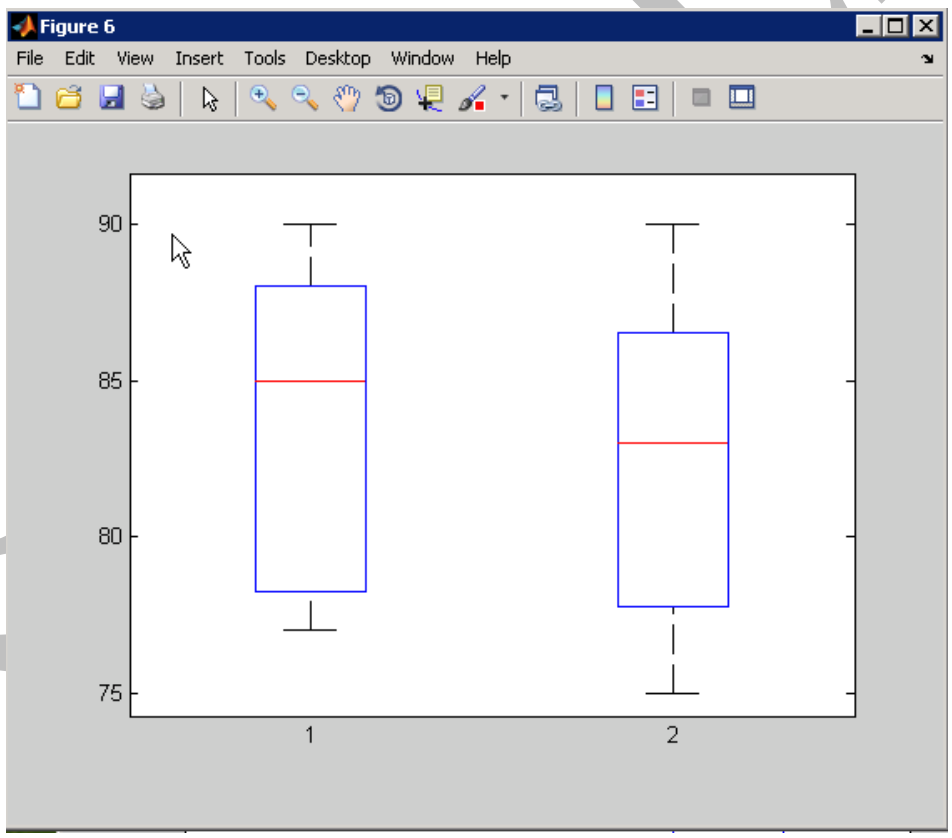
We first begin with the Brown-Forsythe one because this is the only test we checked by making the calculations "by hand" and also with Minitab and R. In MATLAB using the function **vartestn( )** with the corresponding arguments we get:



And the following figures open:



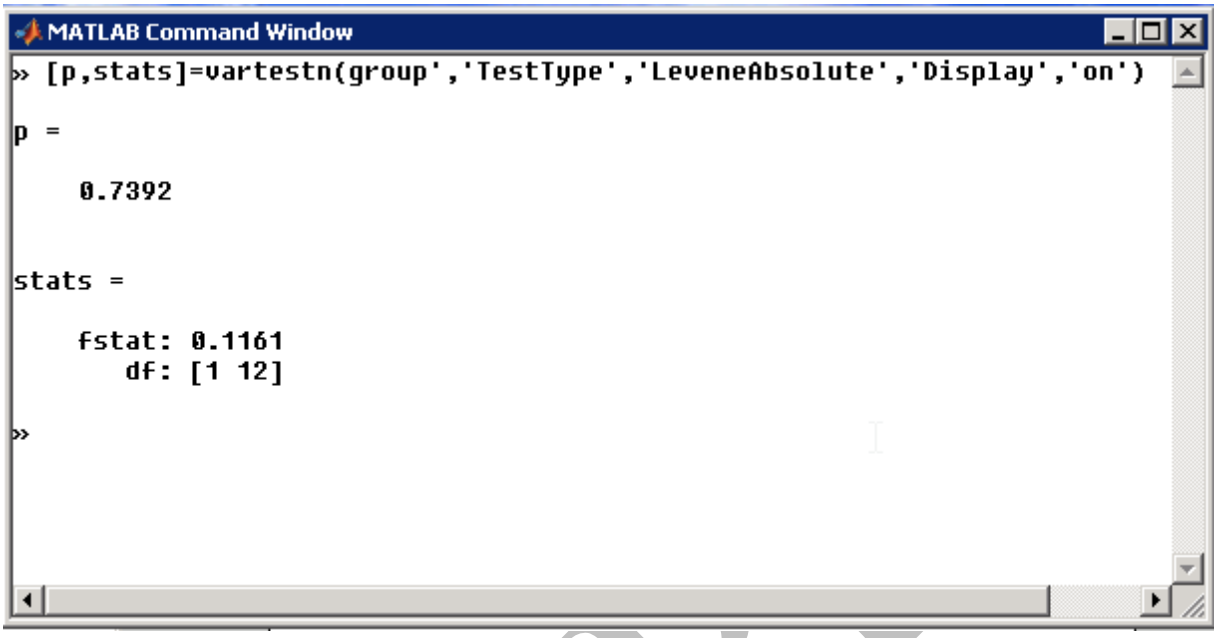
And also the following chart:



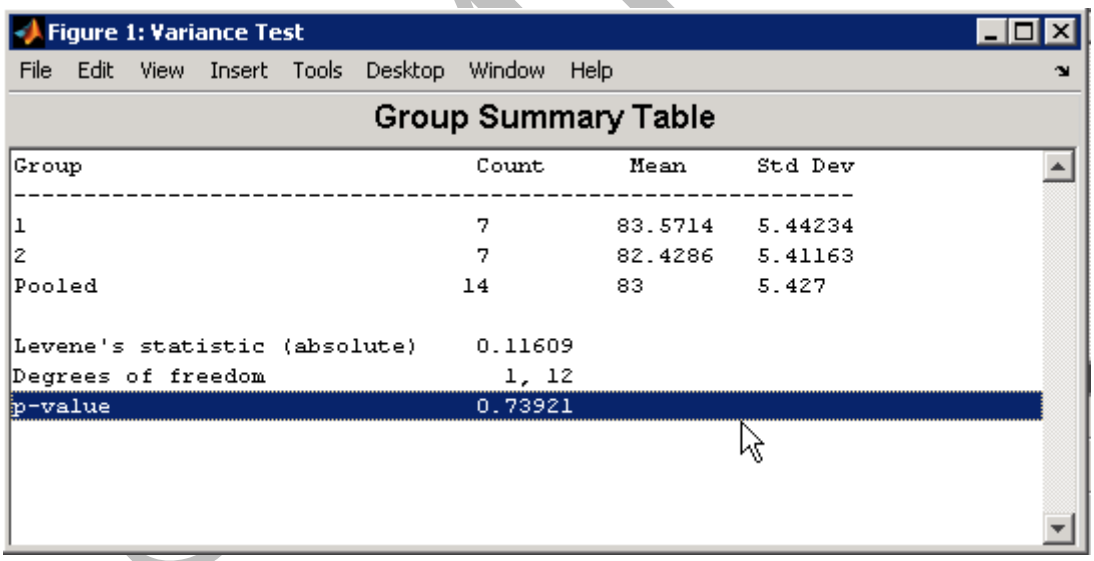
We get exactly the same results as the one calculated by hand and with Minitab and R.

Absolute Levene

And is the Levene absolute test available in MATLAB? Yes:

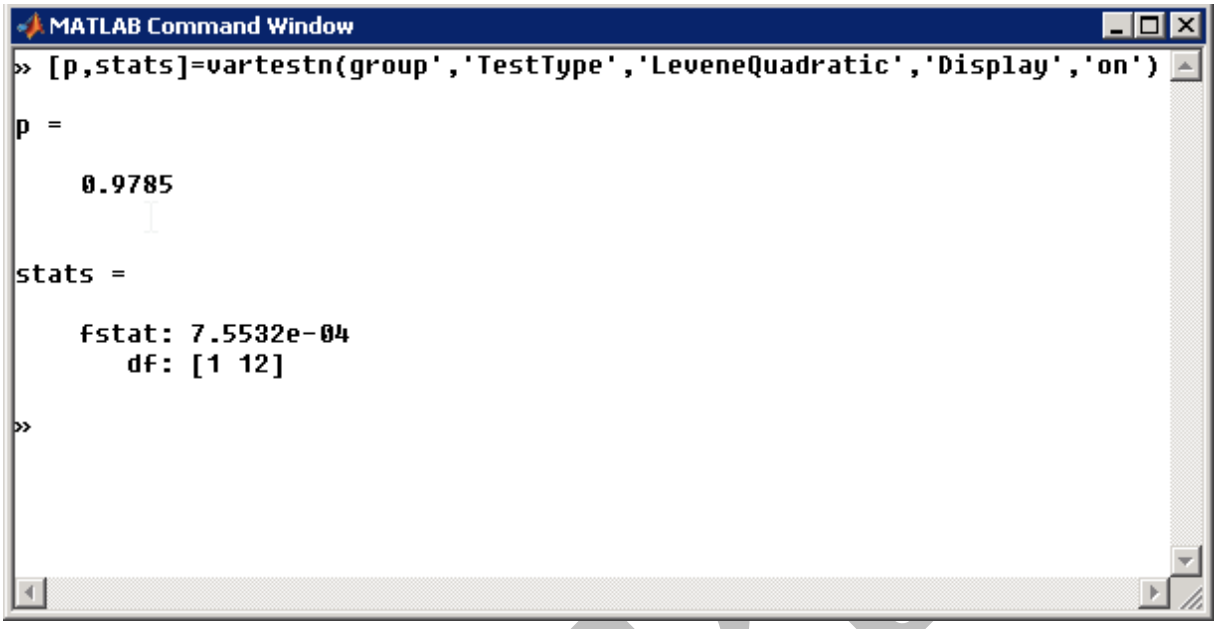


So we get:

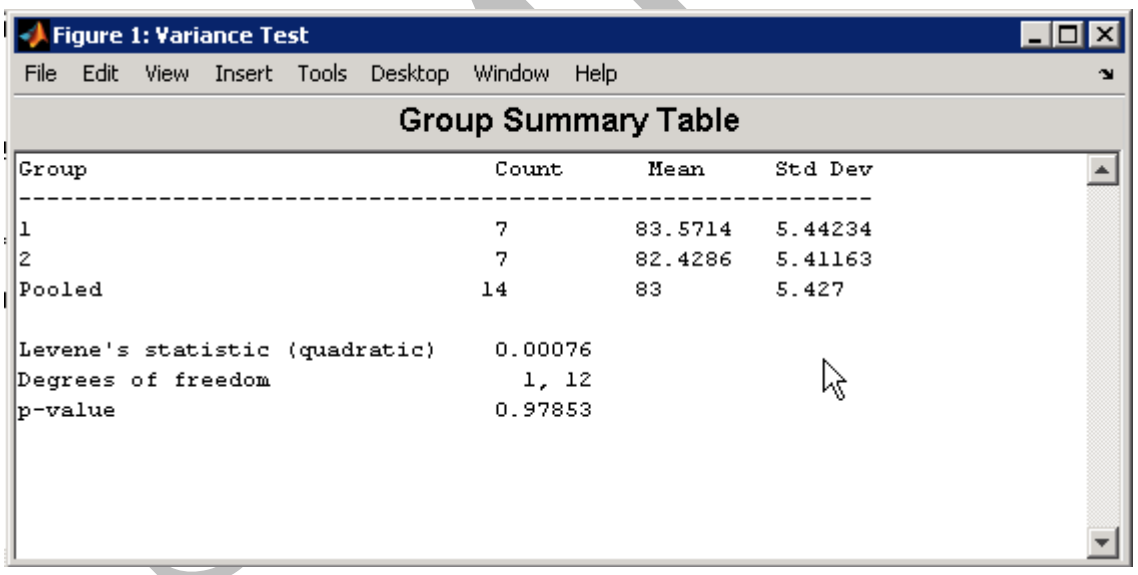


Quadratic Levene (square)

And what about the Leven square test? Yes we have it:



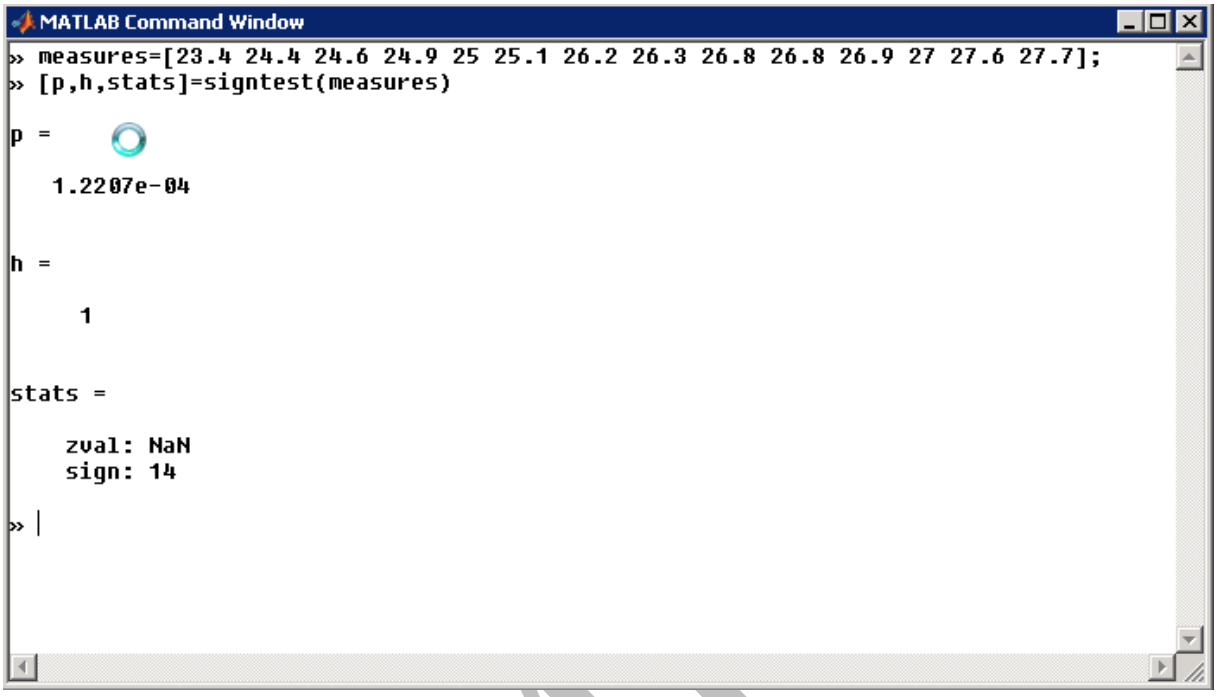
This give us also as output:



We can conclude here that at the opposite of Minitab and R we have much more THV tests available in MATLAB.

Test for the median (one sample sign test)

Now have a look to a very important stuff: the median test with it's confidence interval! For this purpose we will use the **signtest( )** function of MATLAB:



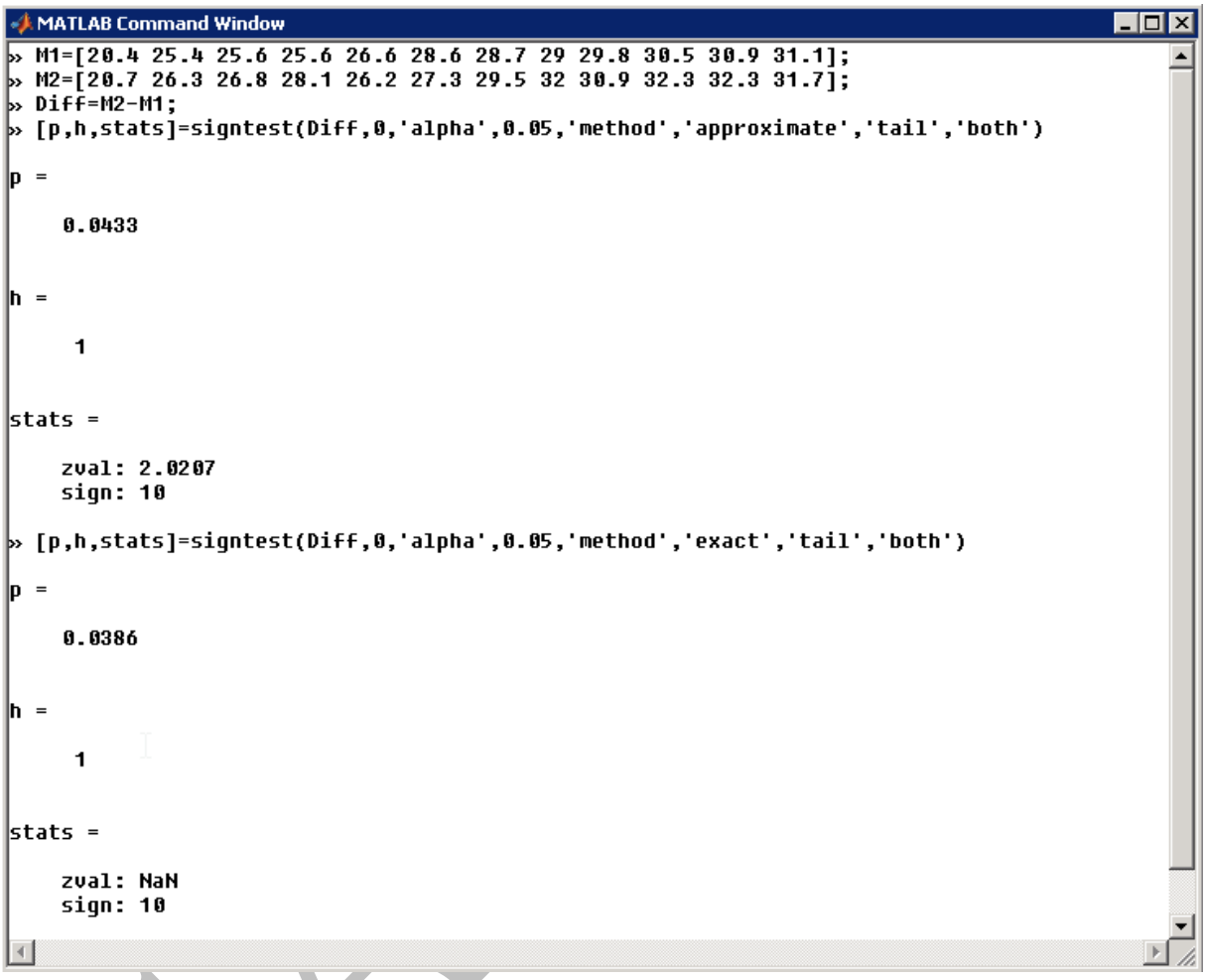
The values are OK and corresponding with those calculated by hand but the very disappointing stuff is that we don't get any confidence interval :-(

For sure, at the opposite of Minitab 15 we get a p-value but at the opposite of R we don't get any C.I.

SAMPLE

Binomial signed rank test (dixit: median test paired samples)

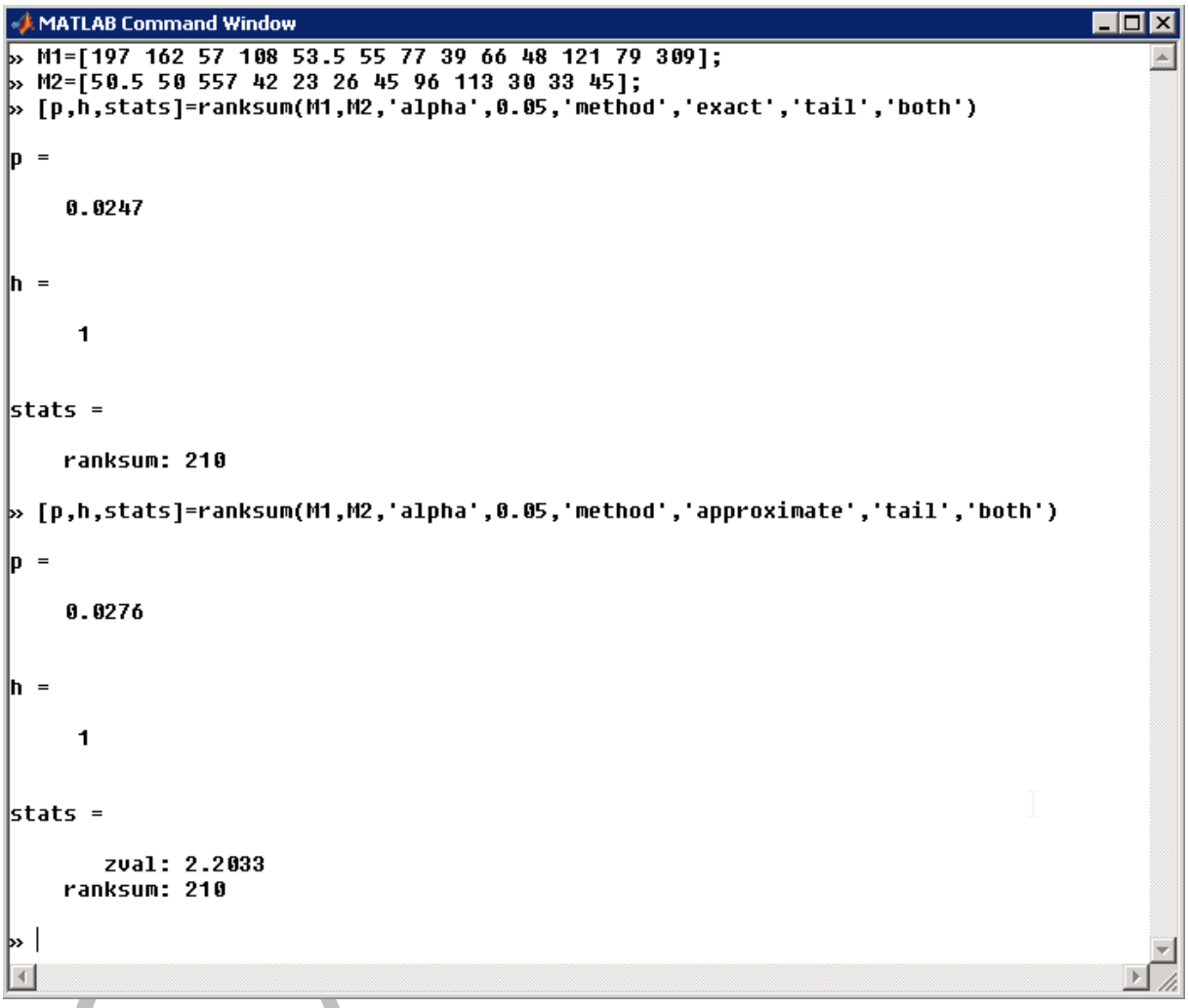
We proved in the theoretical statistic course a possible test form comparing the median of two distributions based on the sign of the differences of the two samples. The purpose is to see now if we get the same result as the one calculated by hand or with Minitab.



As we can see the second result is in perfect adequation with Minitab and thus with the values calculated by hand in the theoretical course. The very disappointing stuff is that we don't get any confidence interval :-(

Two unpaired samples Wilcoxon (Mann-Withney) rank sum test

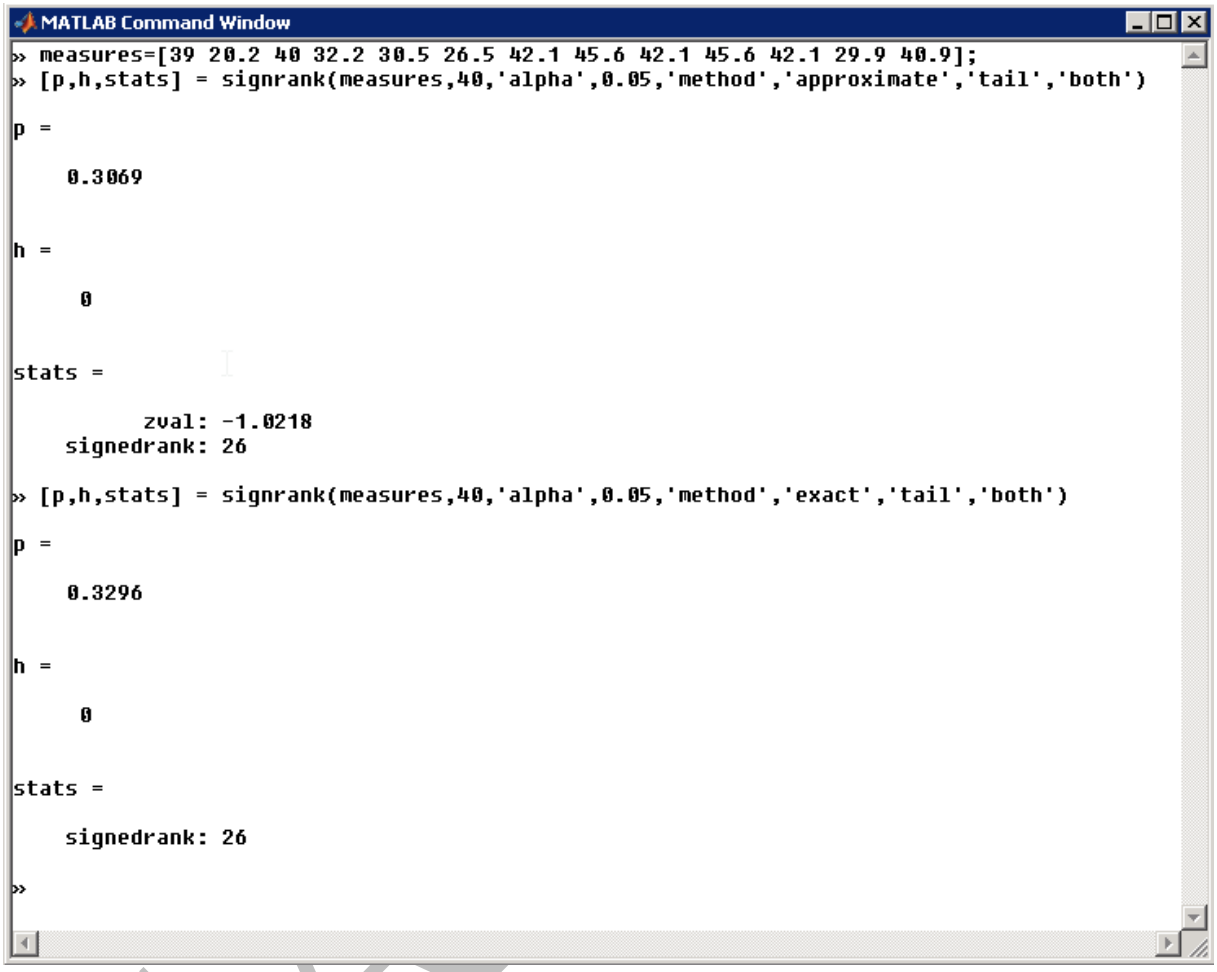
Now the purpose is to see if we found the same result with MATLAB as the one with Minitab and this calculated by hand using the function **ranksum( )**:



We get exactly the same result as with Minitab (for the approximate version) and the same small difference as with the calculation made my hand in the theoretical statistic course.

Wilcoxon signed rank test for 1 sample

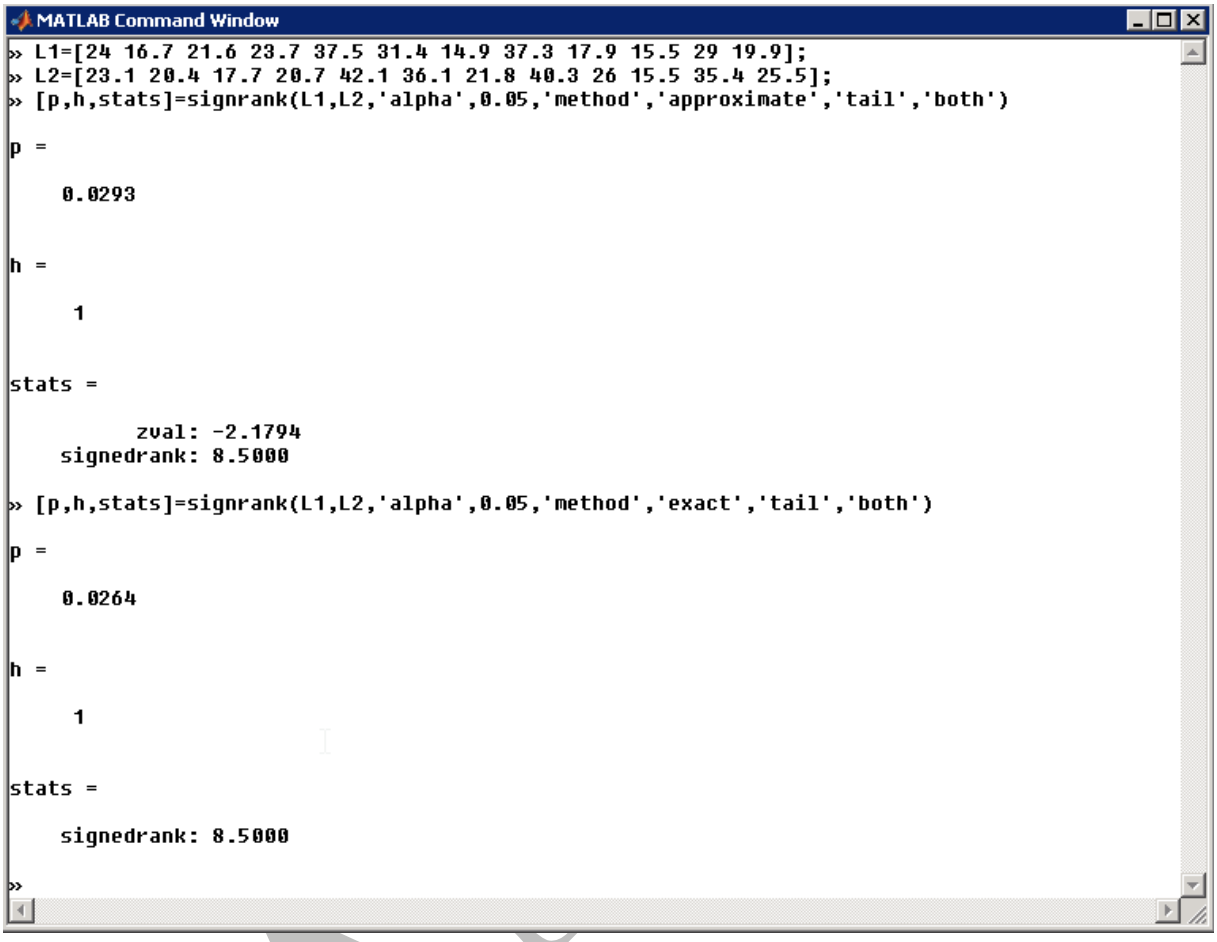
We also proved in the theoretical statistic course another way to make the test about the median using ranks. The Wilcoxon signed rank test is under the function name **signrank( )** in MATLAB:



The exact method corresponds to what we get with Minitab. The approximate method corresponds to what we get in the calculation by hand.

Wilcoxon signed rank test for two paired samples

And what about the Wilcoxon signed rank test for two paired samples compared to the theoretical statistic course



None of the p-values correspond to Minitab or even to R and same thing for the comparison to the values calculate by hand... But the conclusion are the same (we reject the null hypothesis).

ANOVA

ANOVA (ANalysis Of VAriance) is probably on of the biggest subject (if it is not the biggest) in the statistic field. You can found books of more than a thousand of page presenting a lot of different ANOVA models. The purpose here is just to see the ANOVA models that are used and proved at the Master graduate level in most schools worldwide.

One-way fixed factor canonical ANOVA

The one-way fixed factor canonical ANOVA is the simplest existing ANOVA model. The purpose here will be like always to see if we obtain the same results as the one calculated by hand and with Minitab and R.

For this purpose, we will use the function **anova1( )** of MATLAB:

```
MATLAB Command Window
» Team1=[78 88 90 77 85 88 79];
» Team2=[77 75 80 83 87 90 85];
» Team3=[88 86 79 93 79 83 79];
» Teams=[Team1' Team2' Team3']

Teams =

    78    77    88
    88    75    86
    90    80    79
    77    83    93
    85    87    79
    88    90    83
    79    85    79

» [p,table,stats]=anova1(Teams)

p =

    0.8739

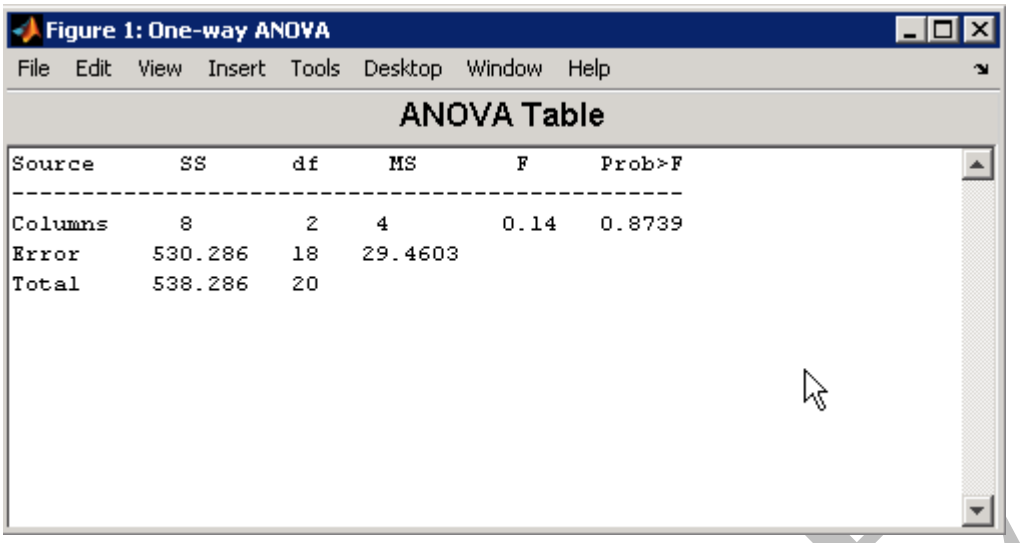

table =

    'Source'      'SS'          'df'    'MS'          'F'          'Prob>F'
    'Columns'     [       8]    [ 2]    [      4]    [0.1358]     [0.8739]
    'Error'       [530.2857]    [18]    [29.4603]          []           []
    'Total'       [538.2857]    [20]           []          []           []

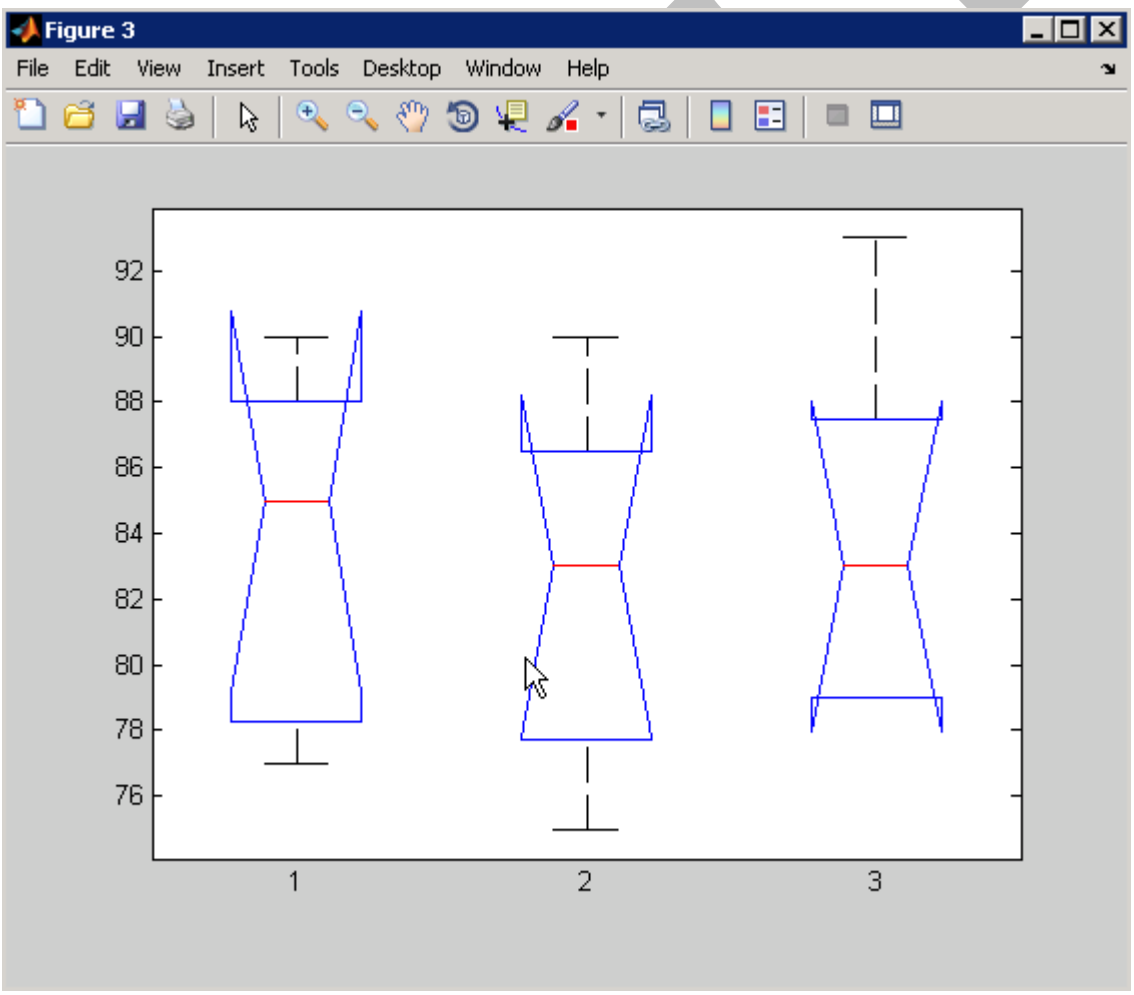

stats =

    gnames: [3x1 char]
         n: [7 7 7]
    source: 'anova1'
     means: [83.5714 82.4286 83.8571]
        df: 18
         s: 5.4277
```

This window opens automatically:

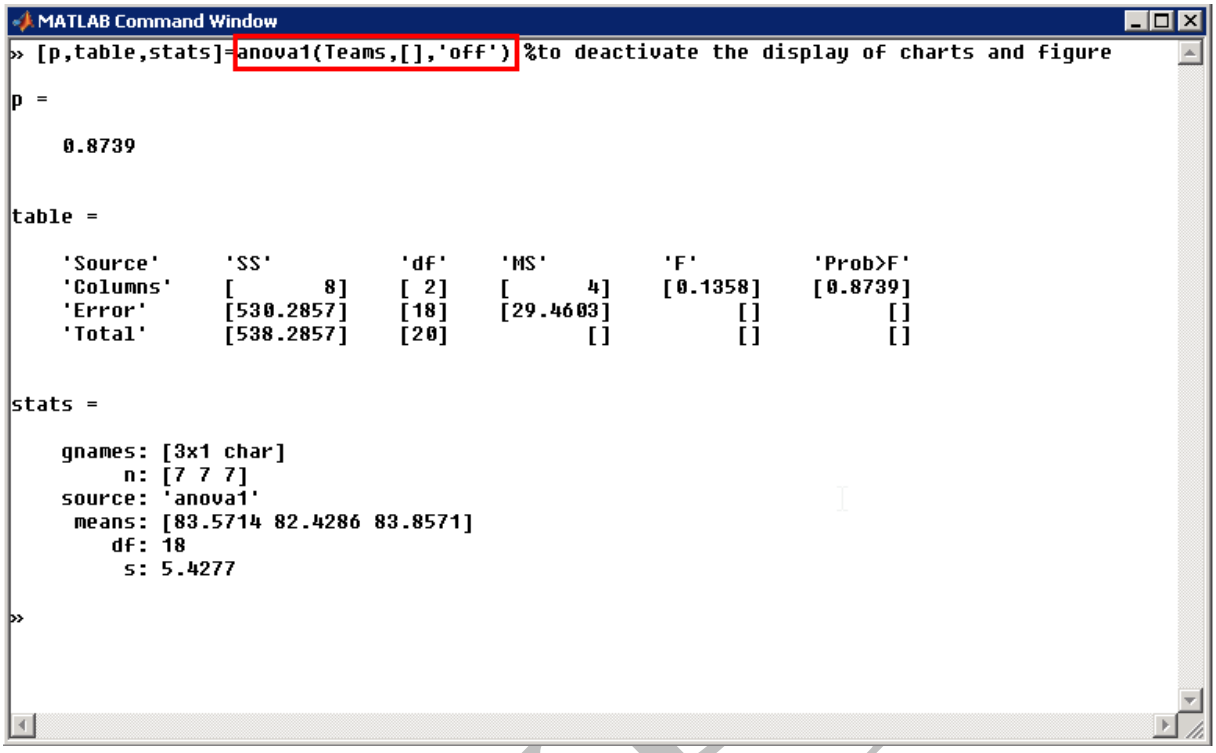


And also this box-plot chart opens also automatically:



What we can conclude is that we get exactly the same results as with Minitab, R and the calculations made by hand.

If we don't want the outputs figure we just have to change the command as following:

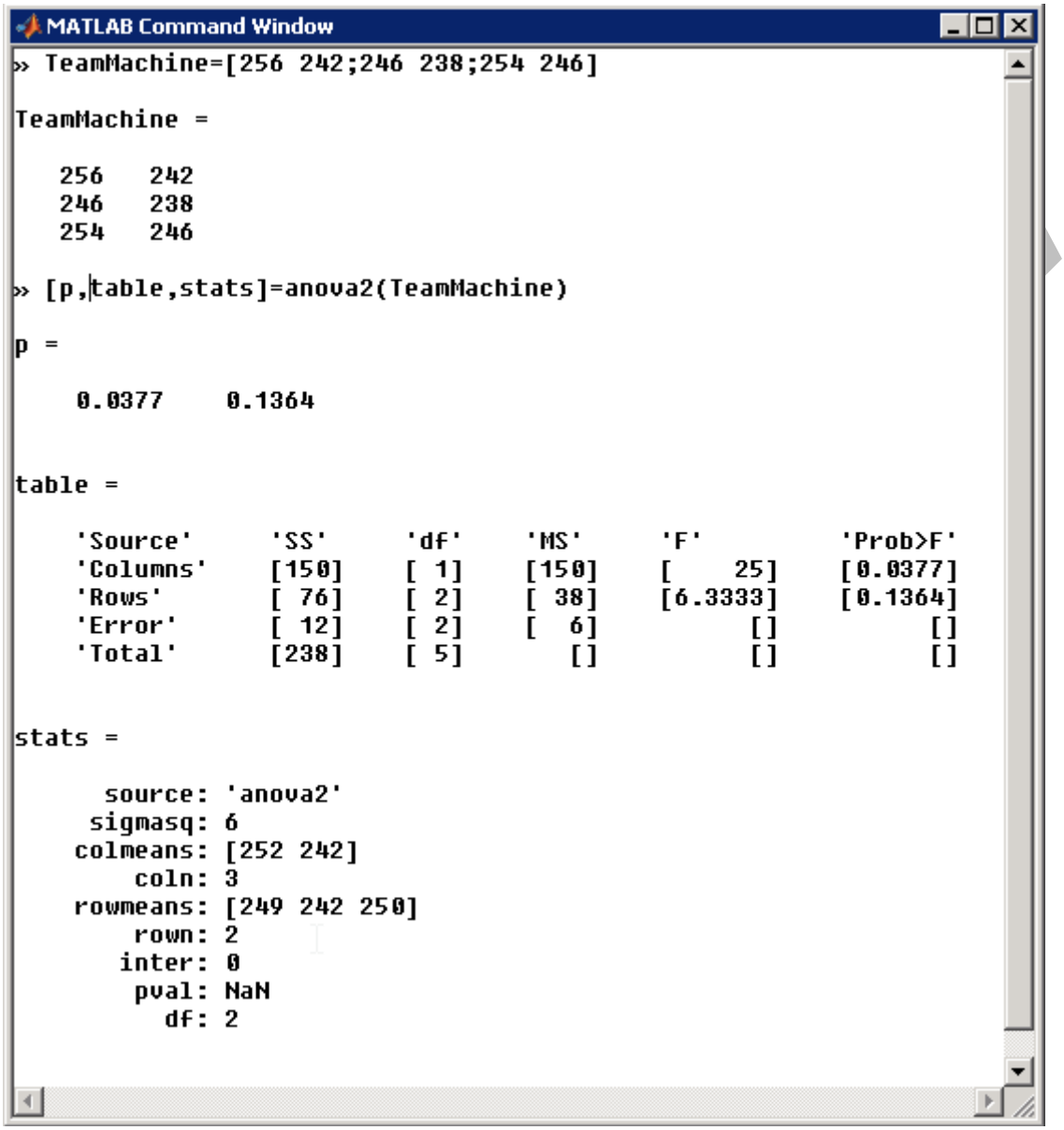


SAMPLE

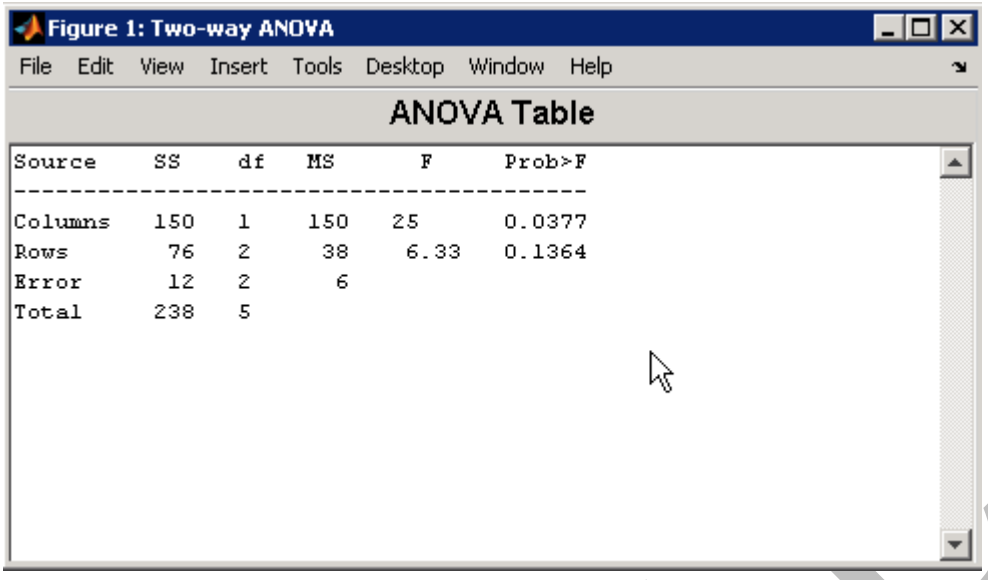
Two-way fixed factors canonical ANOVA

Now have a small look to the two way fixed factors ANOVA without interaction with the small (unrealistic) example we used during the theoretical statistic course to check the correspondence with Minitab, R and the calculations made by hand.

For this purpose we will use the function **anova2( )** of MATLAB:



This window will open automatically:



And so we can see that we get the same result as Minitab, R and the calculations made by hand.

If we want to deactivate the figure we change the function a little bit:

```
MATLAB Command Window
» [p,table,stats]=anova2(TeamMachine,1,'off')

p =

    0.0377    0.1364


table =

    'Source'      'SS'      'df'      'MS'      'F'          'Prob>F'
    'Columns'     [150]     [ 1]      [150]     [    25]     [0.0377]
    'Rows'        [ 76]     [ 2]      [ 38]     [6.3333]     [0.1364]
    'Error'       [ 12]     [ 2]      [  6]          []           []
    'Total'       [238]     [ 5]         []          []           []

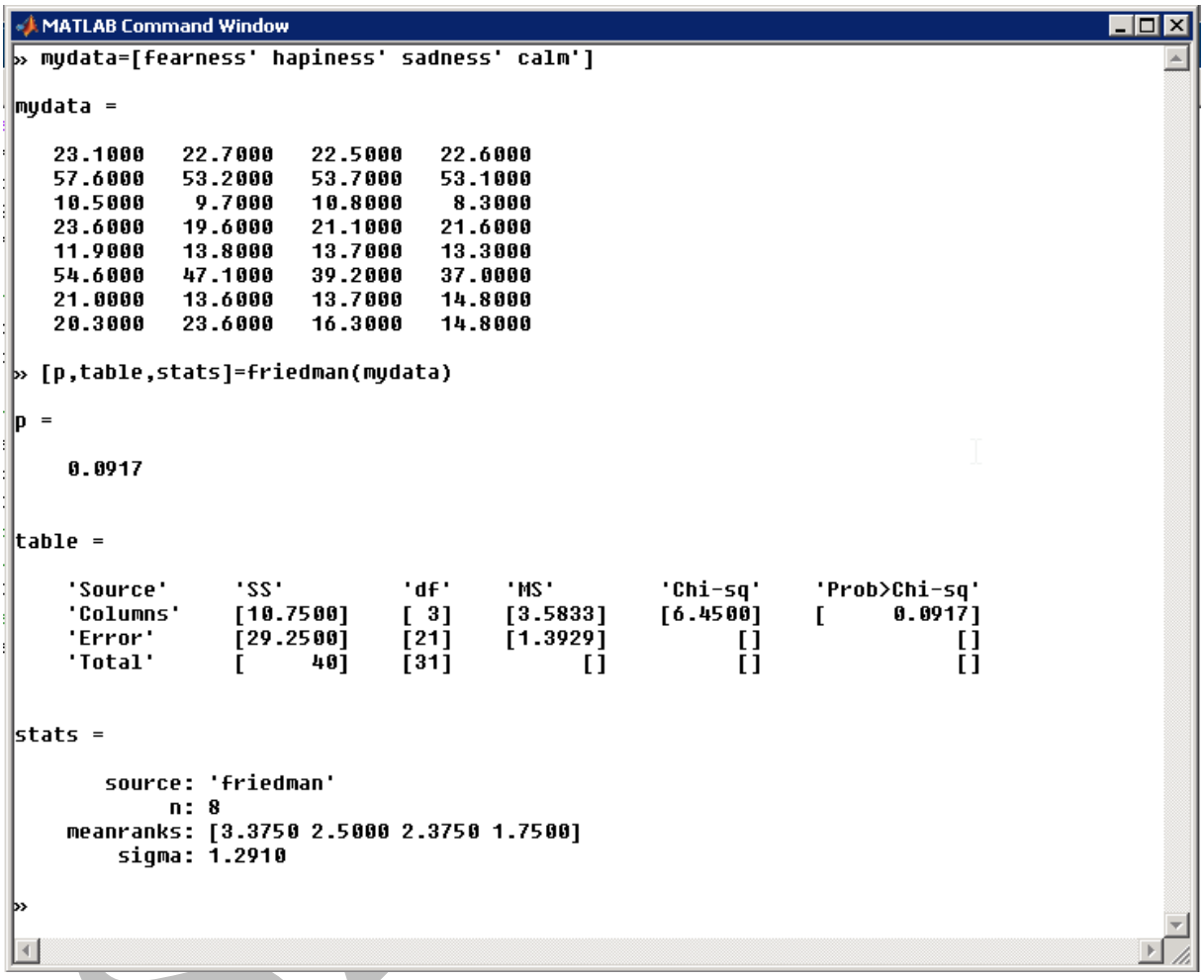

stats =

      source: 'anova2'
     sigmasq: 6
    colmeans: [252 242]
        coln: 3
    rowmeans: [249 242 250]
        rown: 2
```

Friedman's test (canonical non-parametric ANOVA based on ranks)

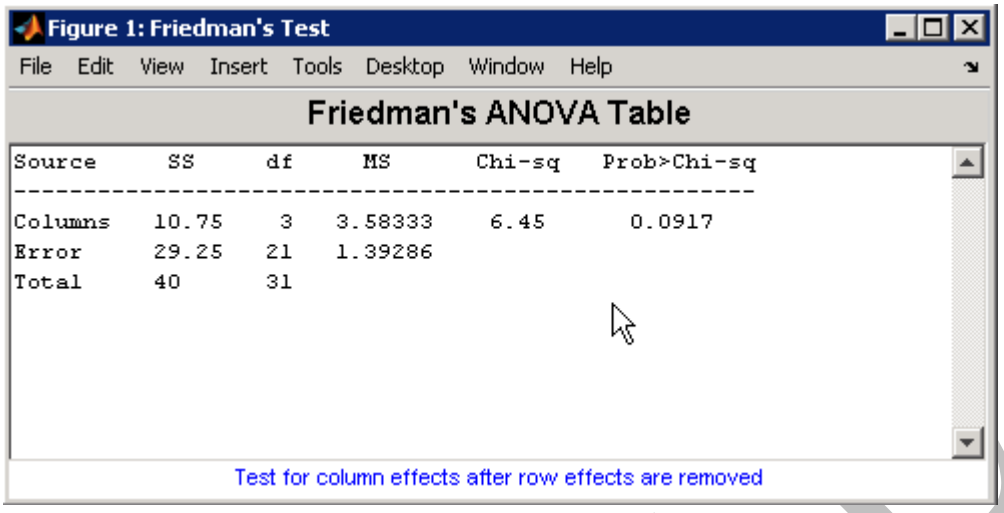
We saw in the theoretical statistic course that the parametric ANOVA can sometimes not be robust to non-normality. For this purpose we have pseudo-proved a way to develop and ranked base non-parametric ANOVA named "Friedman's test".

We will see now using the same datas as the calculations made by hand in Microsoft Excel and also R + Minitab to see if we get the same results using the function **friedman( )**:

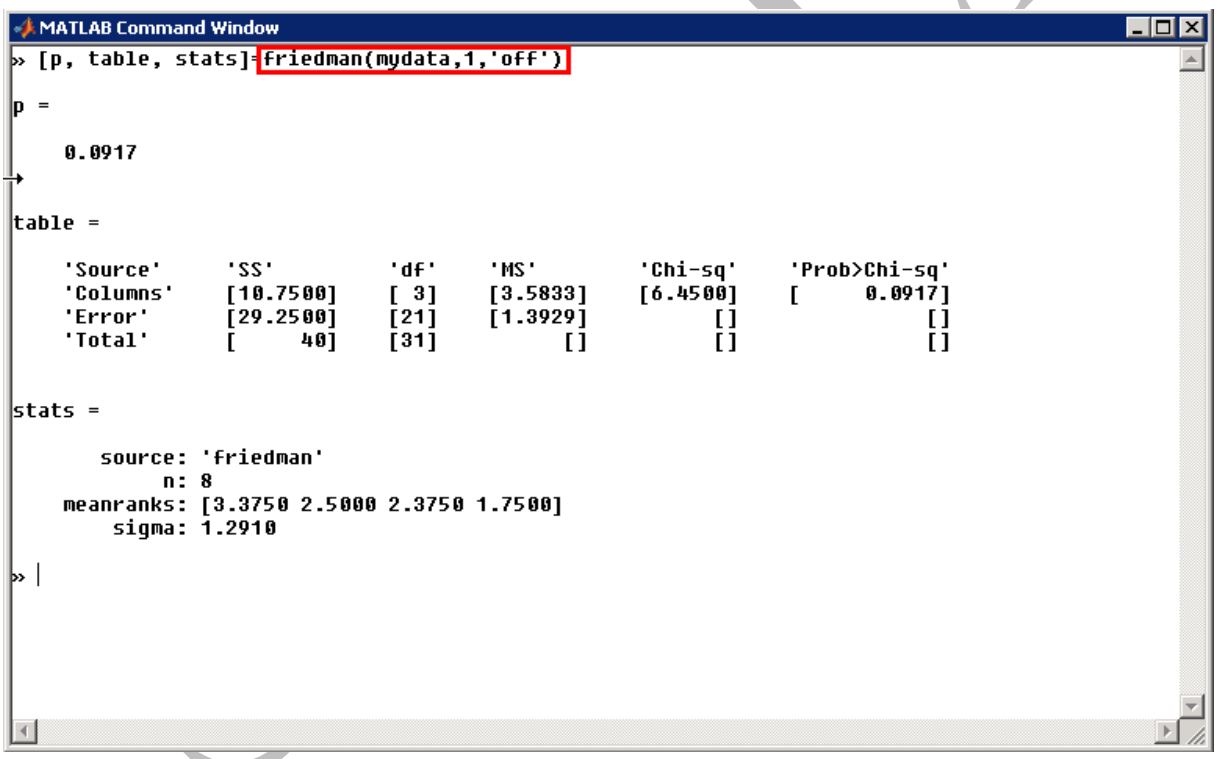


We can see that we get the same values as in theoretical course and more details in comparison of Minitab.

The following windows will also open automatically:



If we don't want to see this figure we just have to change the way we use the function:



Kruskal-Wallis test (canonical non-parametric one-way ANOVA

Once again we will use the original data of the publication of Kruskal and Wallis to see if we get the same results as in the theoretical course:

Standard		Modification 1		Modification 2	
Output	Rank	Output	Rank	Output	Rank
340	5	339	4	347	10
345	9	333	2	343	7
330	1	344	8	349	11
342	6			355	12
338	3				
					Sum
n	5		3	4	12
R	24		14	40	78
R^2/n	115.2		65.333	400.	580.533

Checks : $\sum n = N = 12$ $\qquad \sum R = \frac{1}{2}N(N+1) = 78$

$$H = \frac{12 \times 580.533}{12 \times 13} - 3 \times 13 = 5.656 \simeq \chi^2(2) \qquad \textit{from (1.2)}$$

$\Pr[\chi^2(2) \geqq 5.656] = 0.059$ $\qquad$ *from [9] or [13]*

$\Pr[H(5, 4, 3) \geqq 5.656] = 0.049$ $\qquad$ *from Table 6.1*

Thus, using the function **kruskalwallis( )** we have:

```
MATLAB Command Window
» Machine={'Stand' 'Stand' 'Stand' 'Stand' 'Stand' ...
 'Modif1' 'Modif1' 'Modif1' 'Modif2' 'Modif2' 'Modif2' 'Modif2'};
» Production=[340 345 330 342 338 339 333 344 347 343 349 355];
» [p,tbl,stats] = kruskalwallis(Production,Machine)

p =

    0.0591


tbl =

    'Source'    'SS'          'df'    'MS'          'Chi-sq'    'Prob>Chi-sq'
    'Groups'    [73.5333]     [ 2]    [36.7667]     [5.6564]    [     0.0591]
    'Error'     [69.4667]     [ 9]    [ 7.7185]           []                []
    'Total'     [    143]     [11]            []          []                []

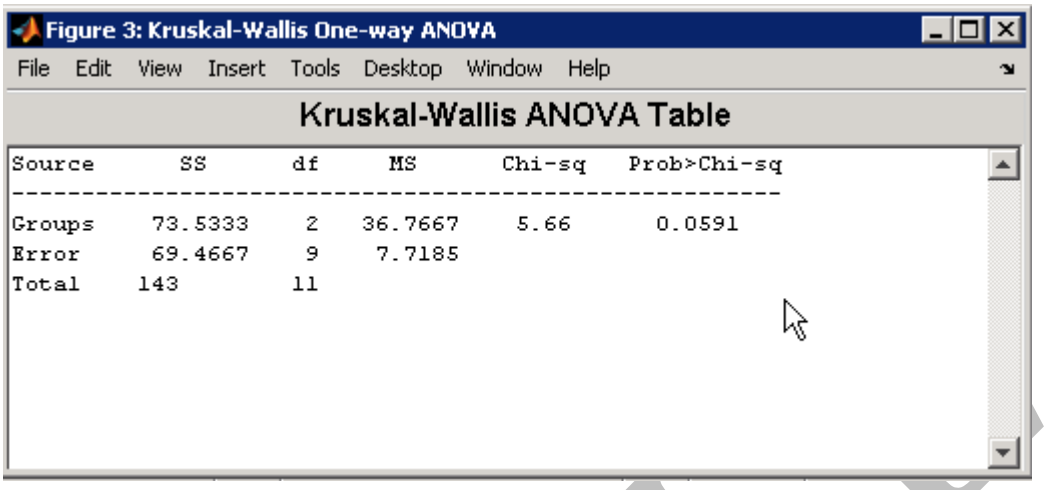

stats =

       gnames: {3x1 cell}
            n: [5 3 4]
       source: 'kruskalwallis'
    meanranks: [4.8000 4.6667 10]
         sumt: 0

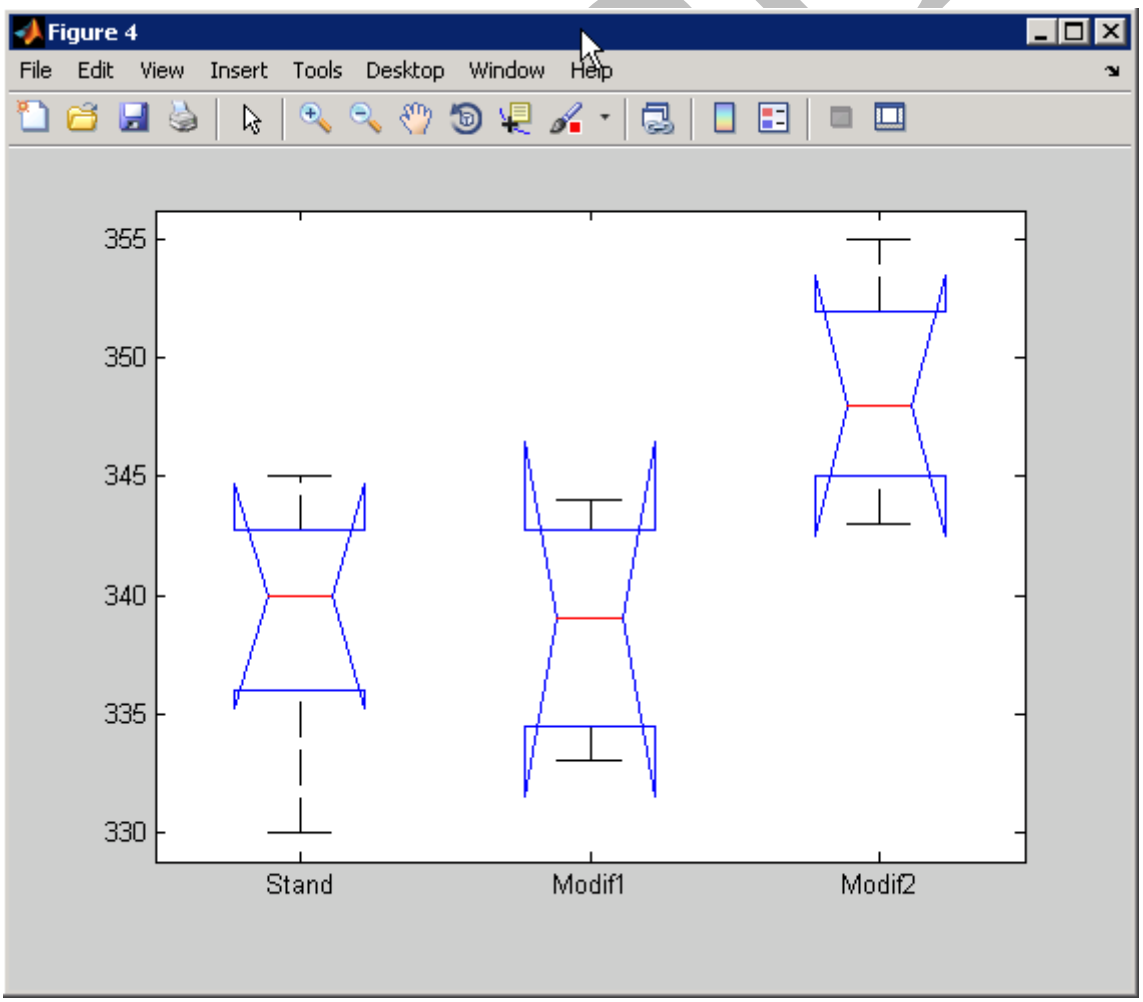
»
```

As we can see it, we get the same results as the theoretical one and as Minitab and R.

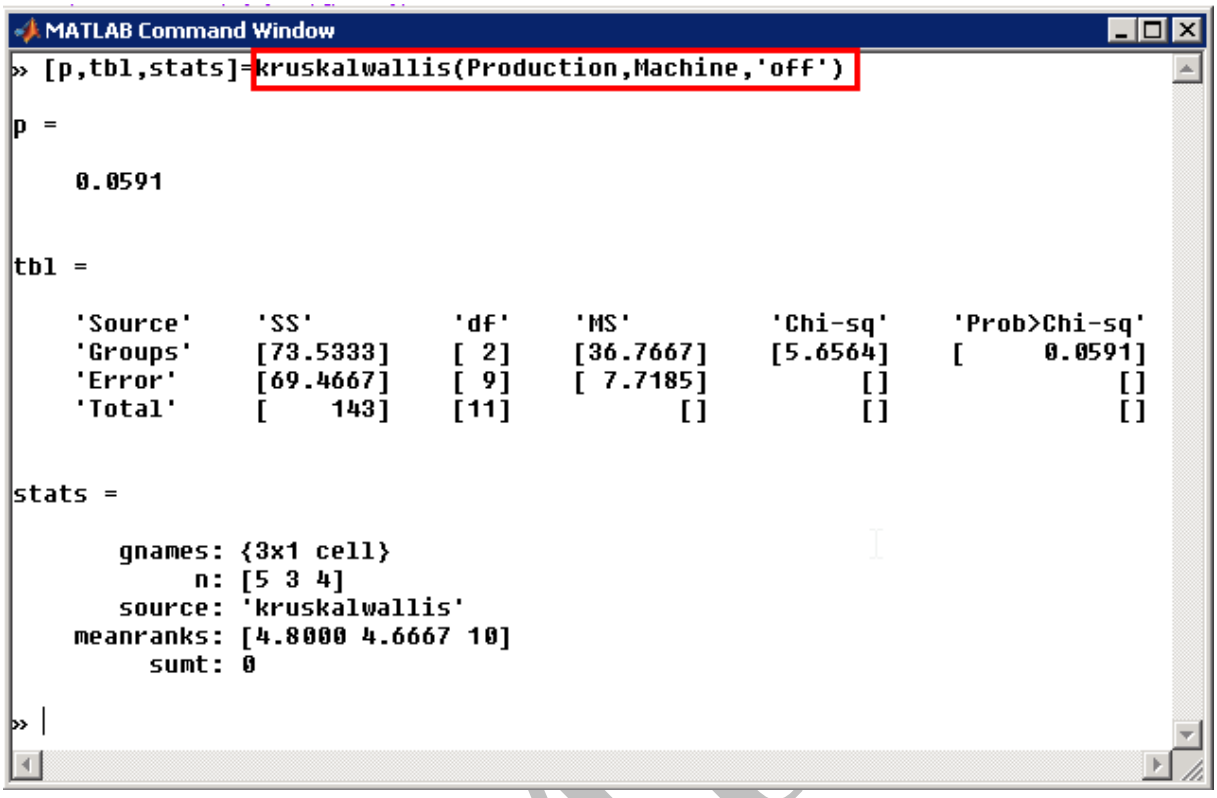
We also have the following figure that will open automatically:



And the following chart:



If we don't want this two previous windows we just have to add a parameter to the function as following:



SAMPLE

Adequation tests

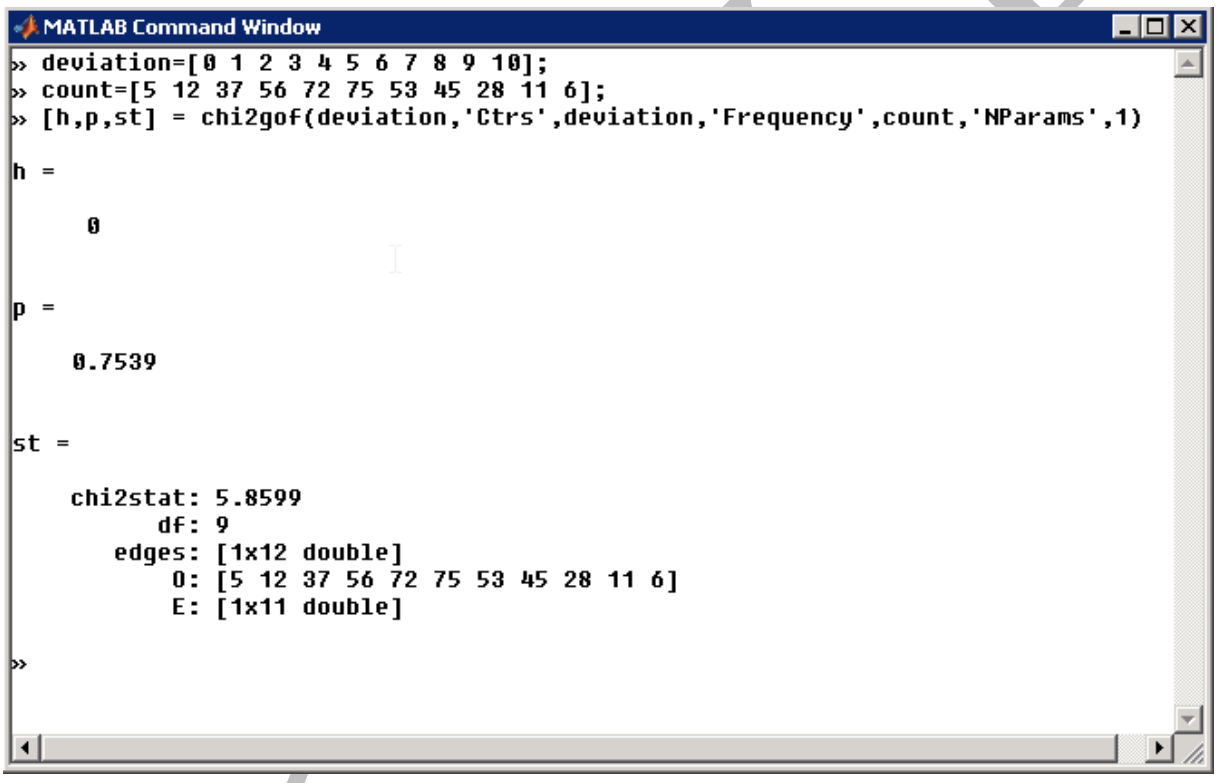
We have seen the important of adequation test in the theoretical course. For its price MATLAB is very poor in possibilities for adequations... for example there is no **native** ranking of best adequation of the forty must well know distributions... Thus I also recommend using the small software of the company:

http://www.mathwave.com

for quick fitting among a lot of theoretical distribution (the remark was the same for Minitab, R and SPSS).

Chi-square goodness of fit for Poisson distribution

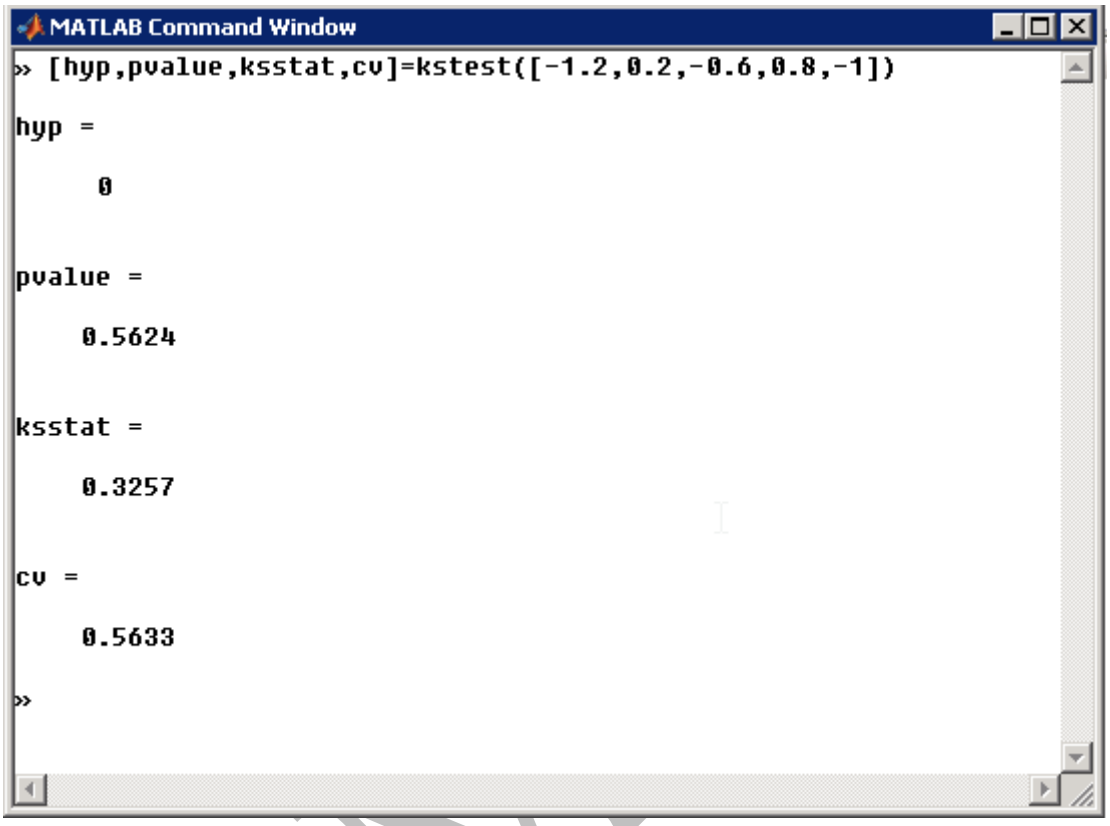
We begin with the first discrete continuous test we proved in the theoretical course using the function **chi2gof( )** and the same values as in theoretical course Minitab and R:



We can see that the output is much more simple as in Minitab and a little bit different (but the conclusion is the same).

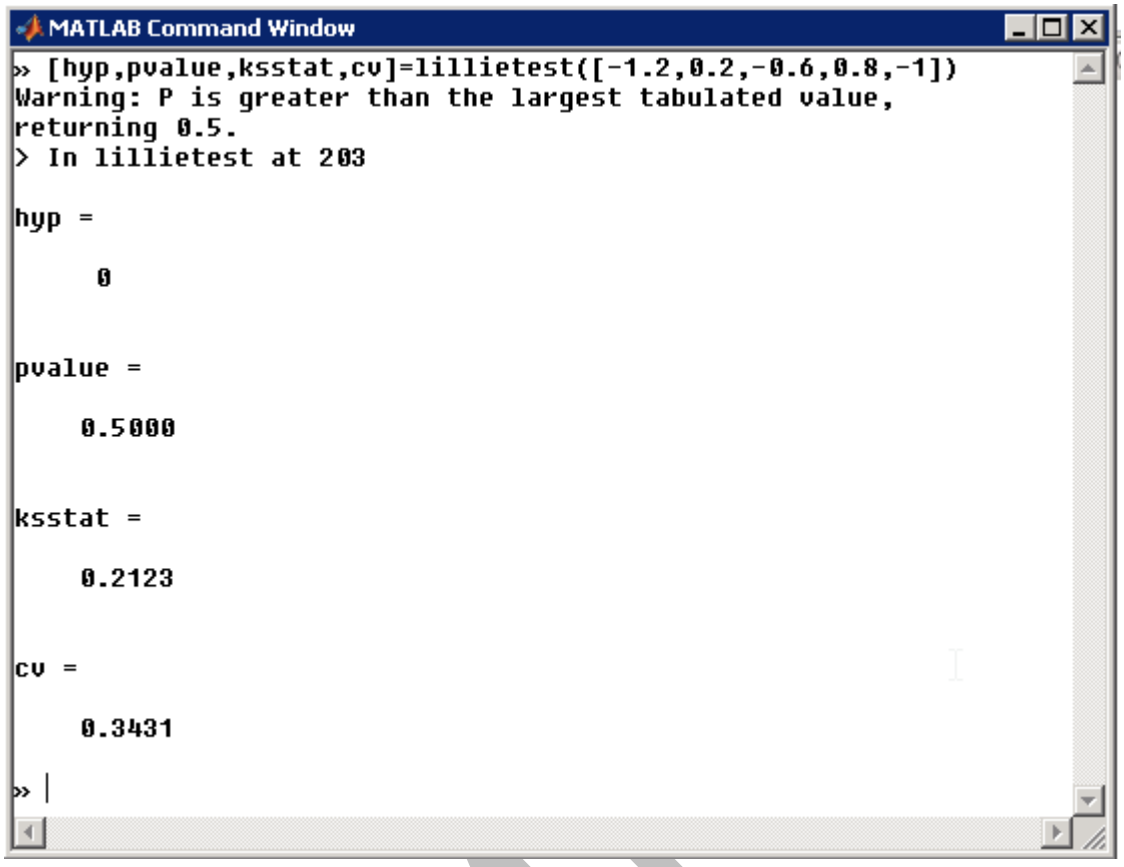
Kolmogorov-Smirnov (one sample) adequation test

Now we take a look to the first adequation continuous test we proved in the theoretical course using the function **kstest( )**:



We get the same distance KS distance and p-value (0.3257) as R and the calculations made by hand but not the same as Minitab. To get the same distance as the one of Minitab we must use the Lilliefor variant with the function **lillietest( )**:

SAM

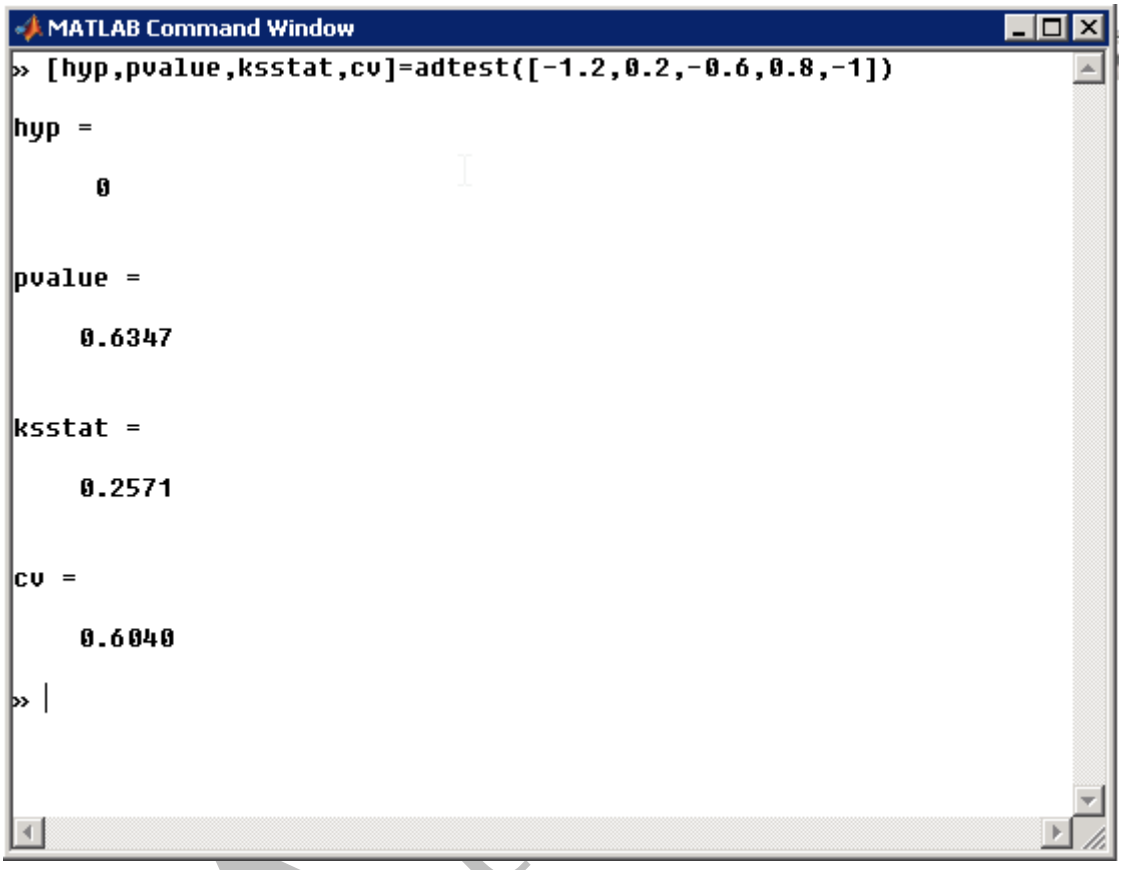


Se we see that now the distance corresponds to the one given by Minitab.

SAMPLE

Anderson-Darling adequation test

Depending on the number of values we have some adequation tests are more powerful than others. Thus it is important to not work only with one adequation test. Another test we proved in the theoretical course is the Anderson-Darling test. Let us see if using the function **adtest( )** we found the same results as in the theoretical statistic course:



We get the same distance (0.2571) as Minitab and the calculations made by hand but not the same as R (0.636). The p-value is not the same as R (0.6058) and nore Minitab (0.535) but we know why...

Regression Techniques

As we know, regression is a subject also as big as the ANOVA. There are books of thousands of pages presenting the most common regression techniques models and even with thousand pages there are not exhaustive.

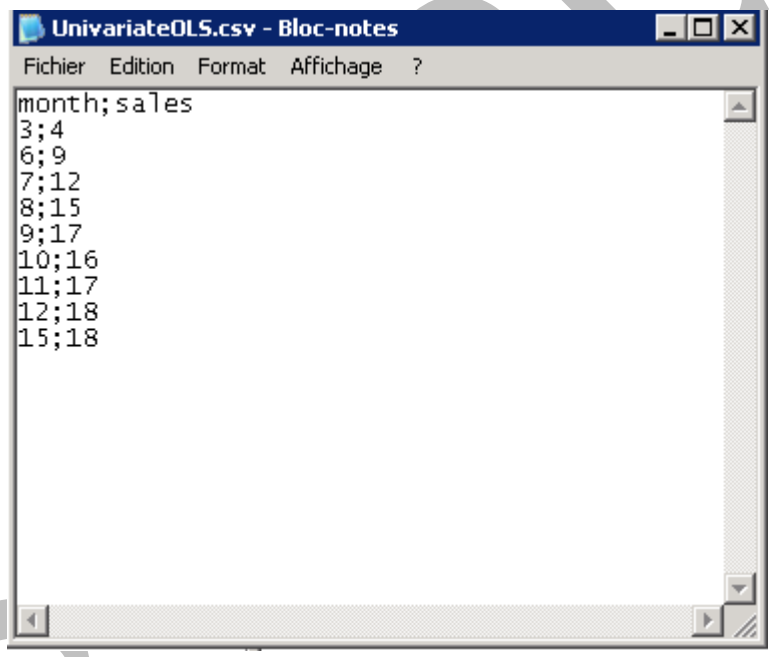
We will first begin by the most simple models in the order we normally study the at school and university.

Ordinary Least Squares (parametric) univariate regression

The parametric Ordinary Least Square regression is a undergraduate technic then it is normal that we begin with this one.

1st method (coefficient and constant cannot be extracted)

There exist multiple ways with the statistical toolbox to make an univariate parametric OLS regression. For this purpose consider the same data as in theoretical course:



By using the function **LinearModel.fit( )** we get the classic univariate coefficients analysis table:

```
Command Window
>> mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
%in 2013a use LinearModel.fit
%in 2013b use fitlm
mdl = LinearModel.fit(mydata,'linear');
>> mdl

mdl =

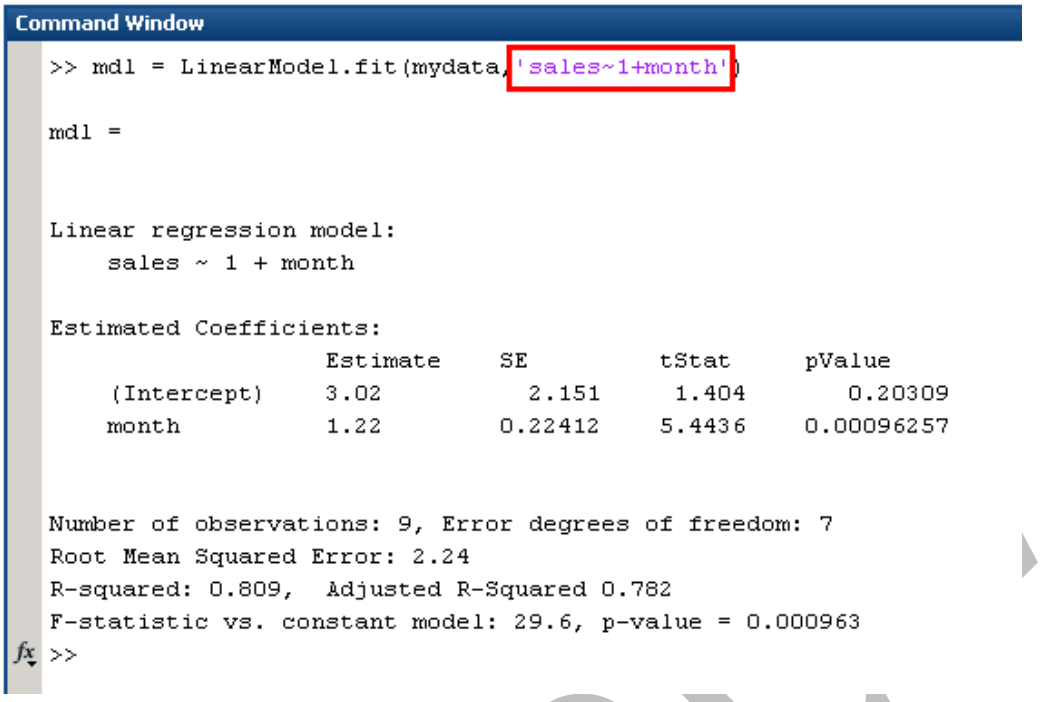

Linear regression model:
    sales ~ 1 + month

Estimated Coefficients:
                   Estimate    SE          tStat     pValue
    (Intercept)    3.02          2.151      1.404       0.20309
    month          1.22        0.22412    5.4436    0.00096257

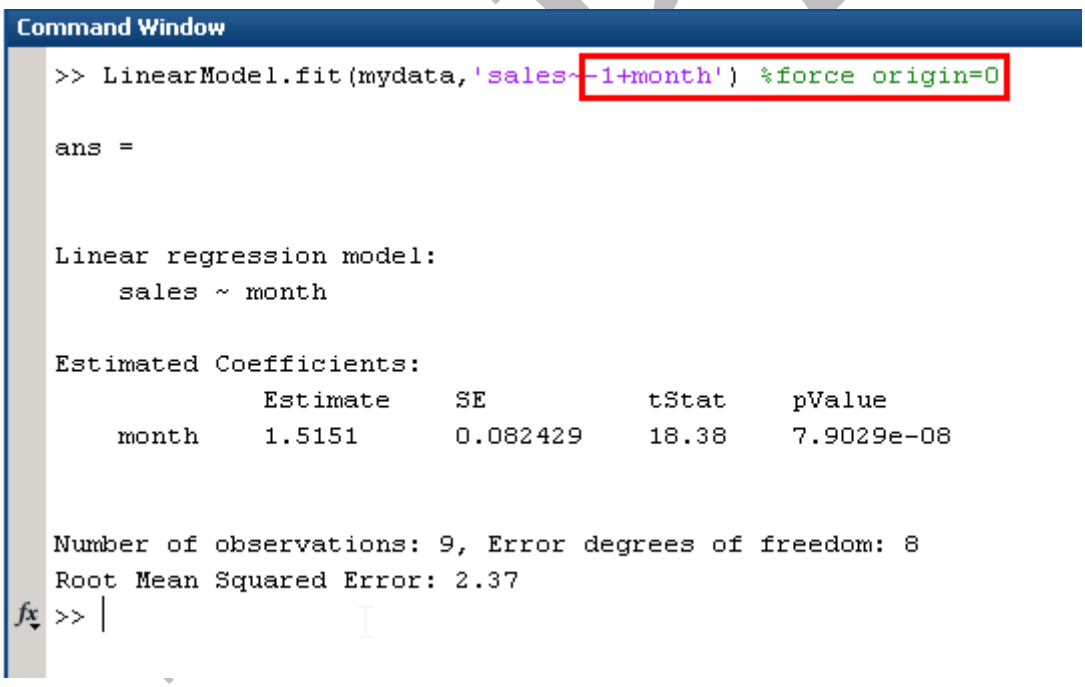

Number of observations: 9, Error degrees of freedom: 7
Root Mean Squared Error: 2.24
R-squared: 0.809,  Adjusted R-Squared 0.782
F-statistic vs. constant model: 29.6, p-value = 0.000963
fx >>
```

All values corresponds with the one obtained in the theoretical course. If we want to specify explicitly the equation of the regression we can do this easily like in R to be able later to build a more complex model:

SAM



Thanks to this notation we can for example easily for the constant to be zero:



We can also get the famous ANOVA table of the regression by using the function **anova( )** with in bracket the variable containing our linear model:

```
Command Window
>> anova(mdl)

ans =

              SumSq     DF     MeanSq     F          pValue
    month     148.84    1      148.84     29.633     0.00096257
    Error      35.16    7      5.0229

>>
```

Or we can also predict values using the model by using the function **predict( )**:

```
Command Window
>> predict(mdl,3)

ans =

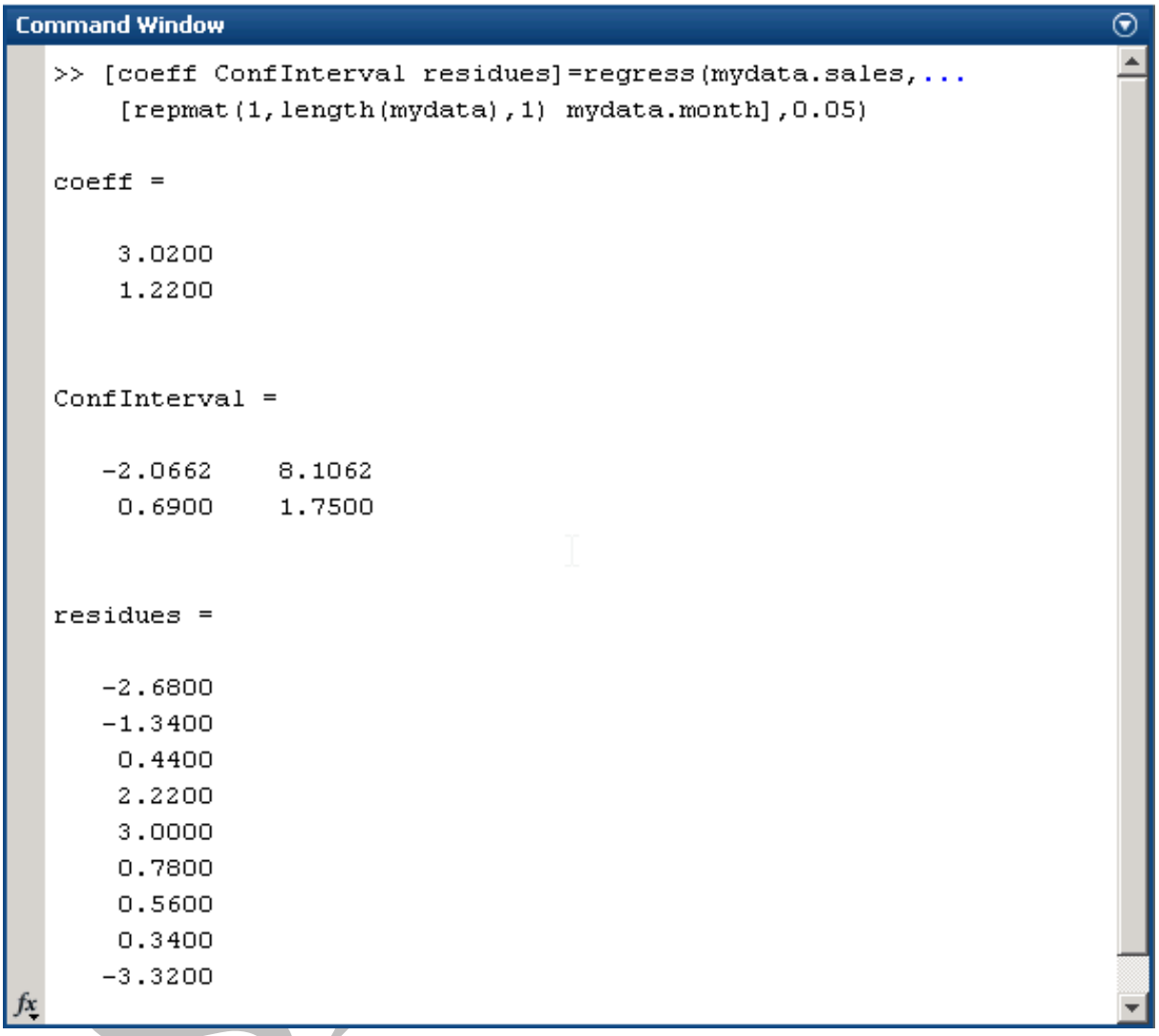
    6.6800

>>
```

SAMPLE

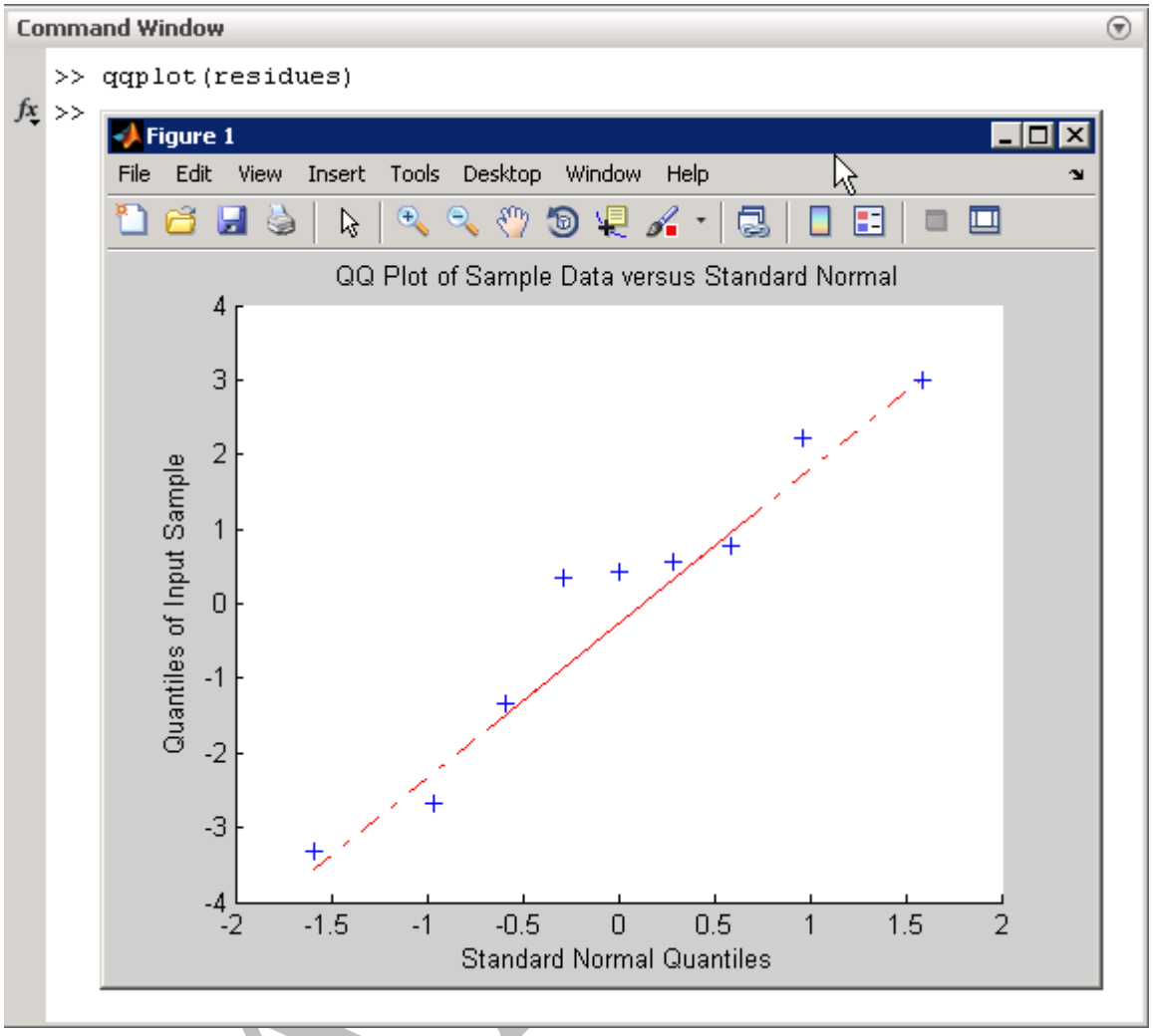
2nd method (coefficient and constant and others can be extracted)

The problem as far as I know with the first method is that you can't extract only the coefficient of the model and this is a major issue in financial application. As workaround I recommend the usage of the function **regress( )** with a syntax that is not intuitive but that in fact simply use the explicit matrix of coefficient that we have seen in the theoretical cours:



So in this way it is very easy as you can see it to extract information of the linear model!

Afterwards if we want to analyze the residues normality in a qualitative way we can do this for example with a simple **qqplot( )**:

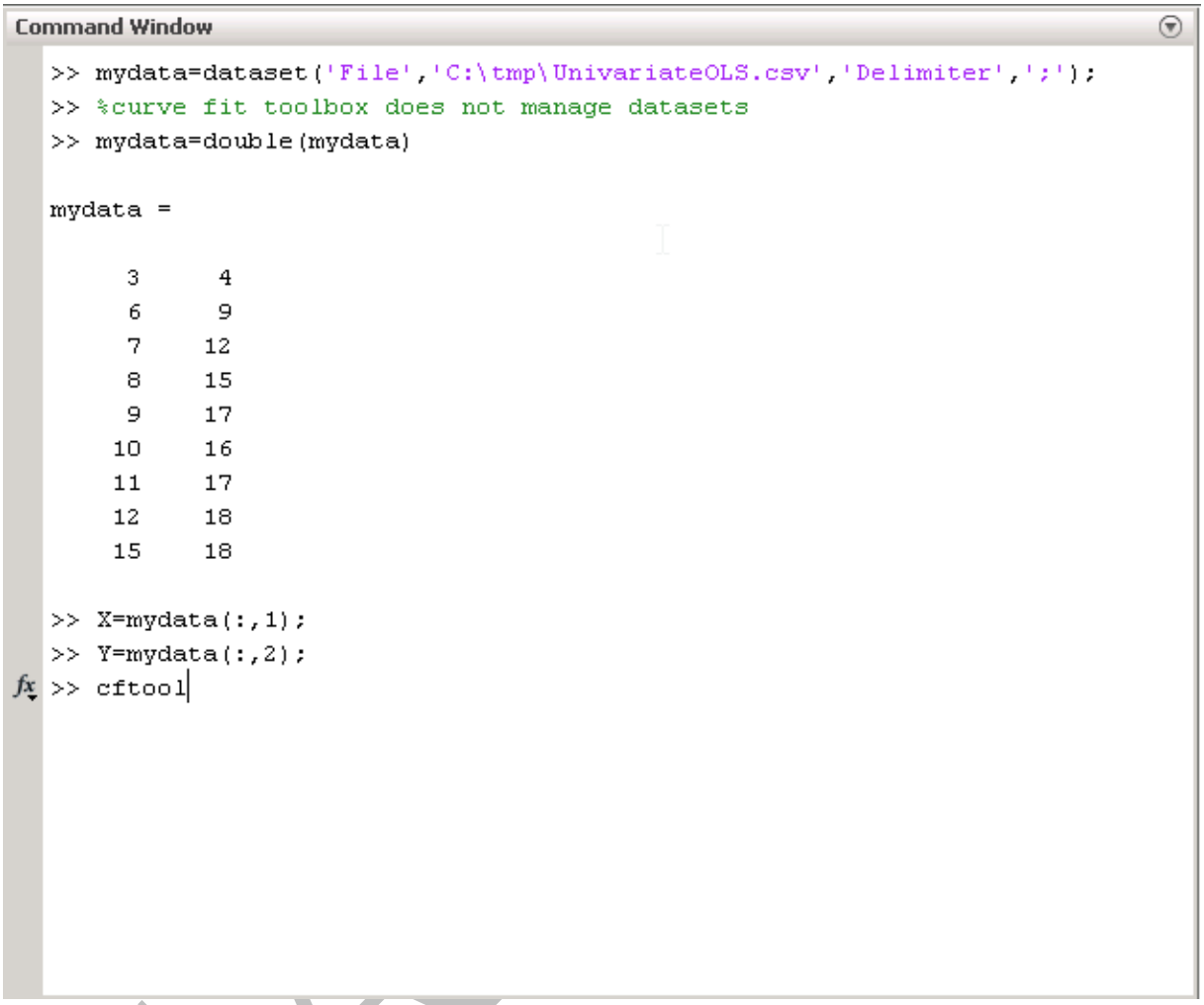


or with an adequation test (see above for this last subject)

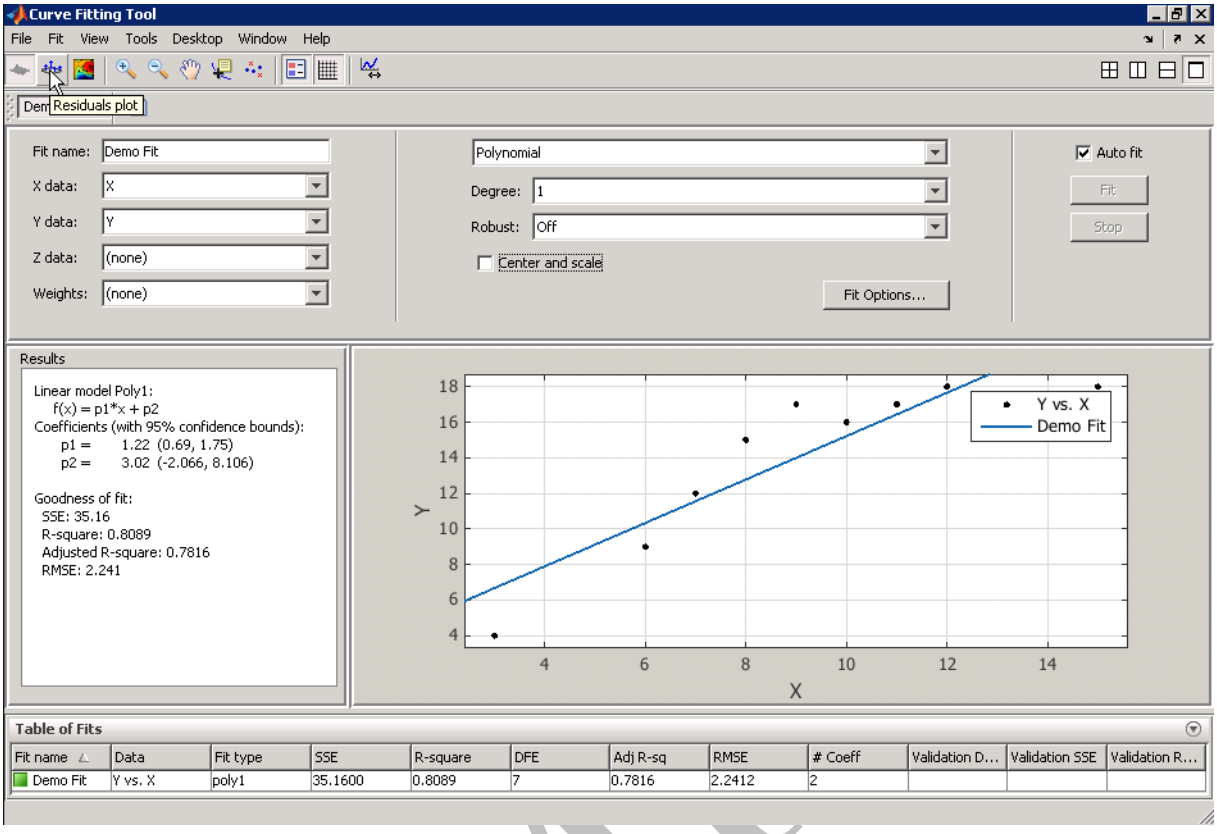


3rd method (using Curve Fitting Toolbox)

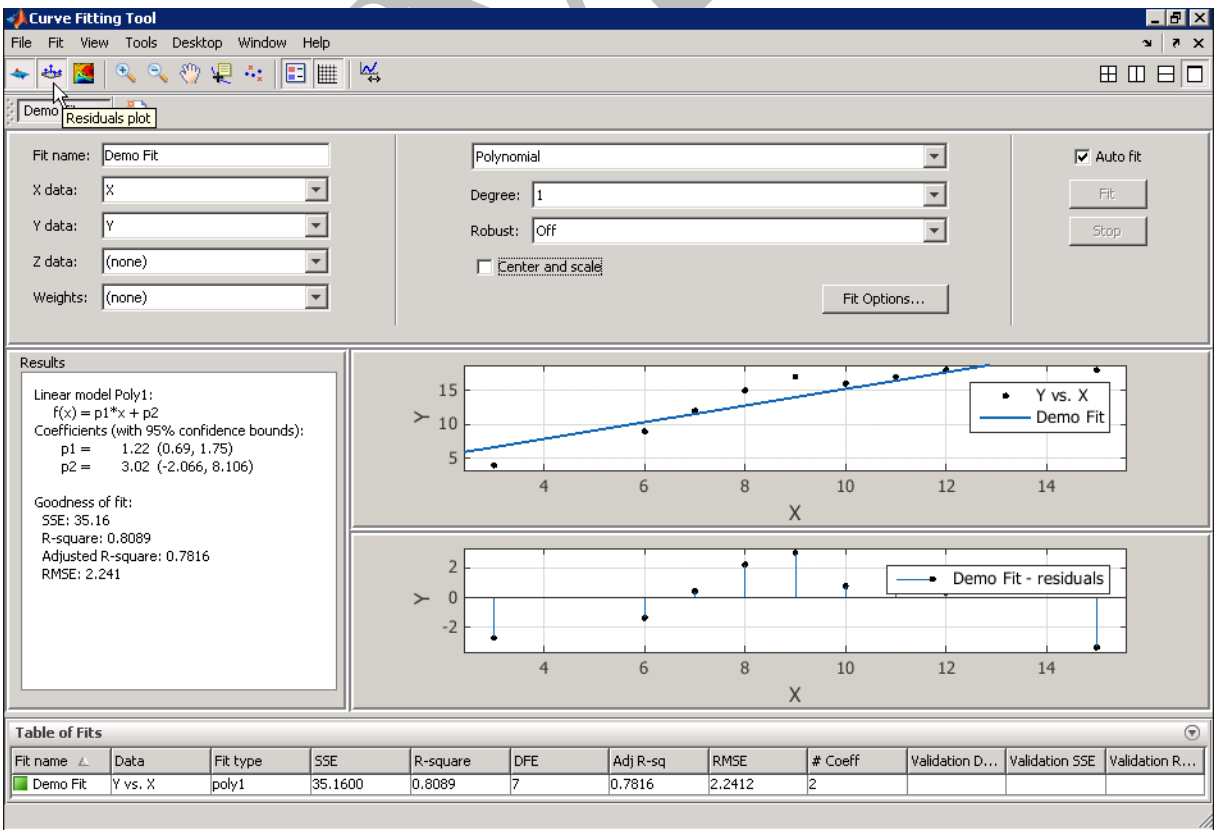
Fort those that doesn't like to use to much MATLAB commands you can use the curve fitting toolbox wizard by typing the command **cftool**:



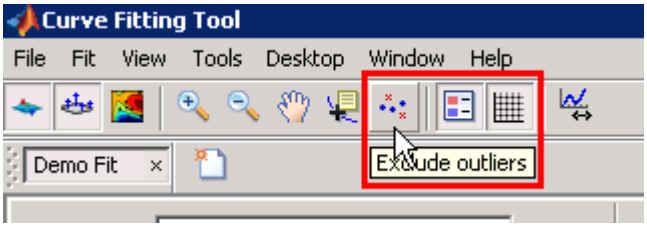
Then you will get:



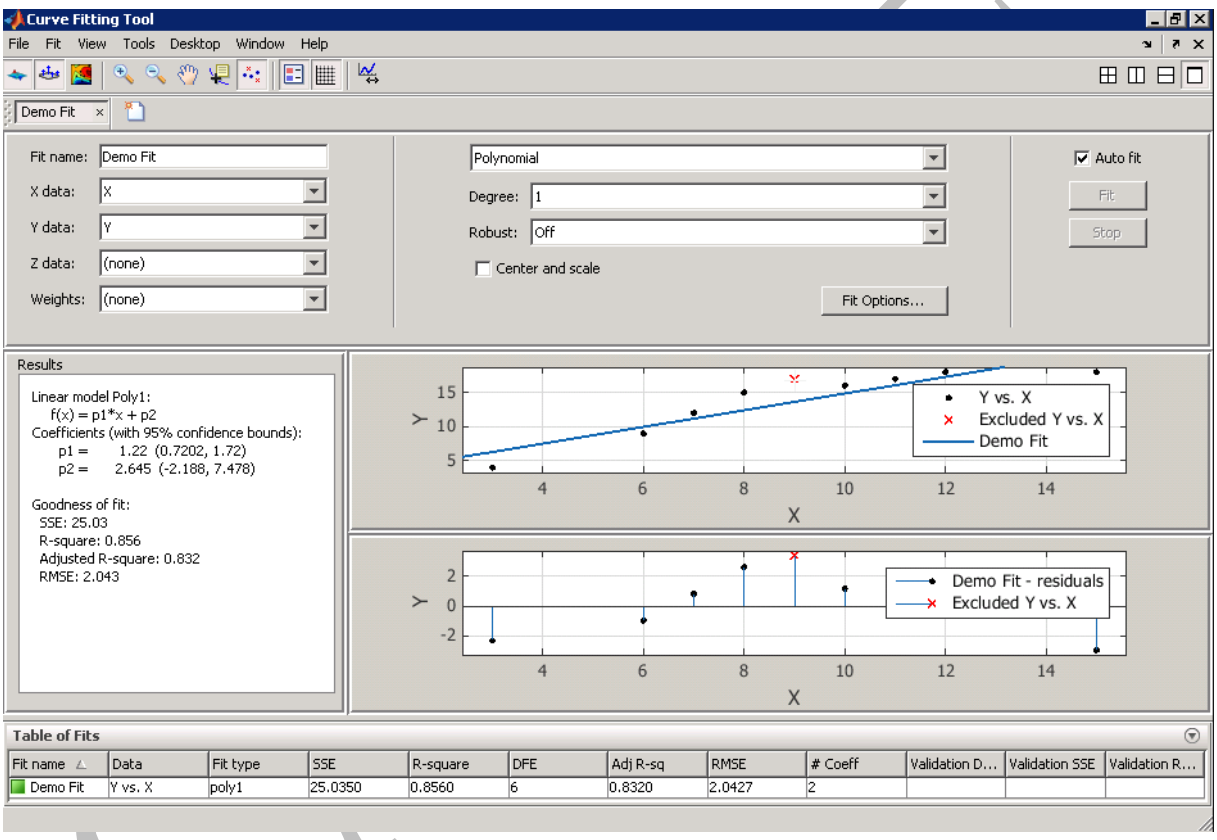
If you click on the button **Residuals plots** you get:



Now a more interesting stuff to focus on is the management of outliers. If you click on the button **Exclude outliers**:

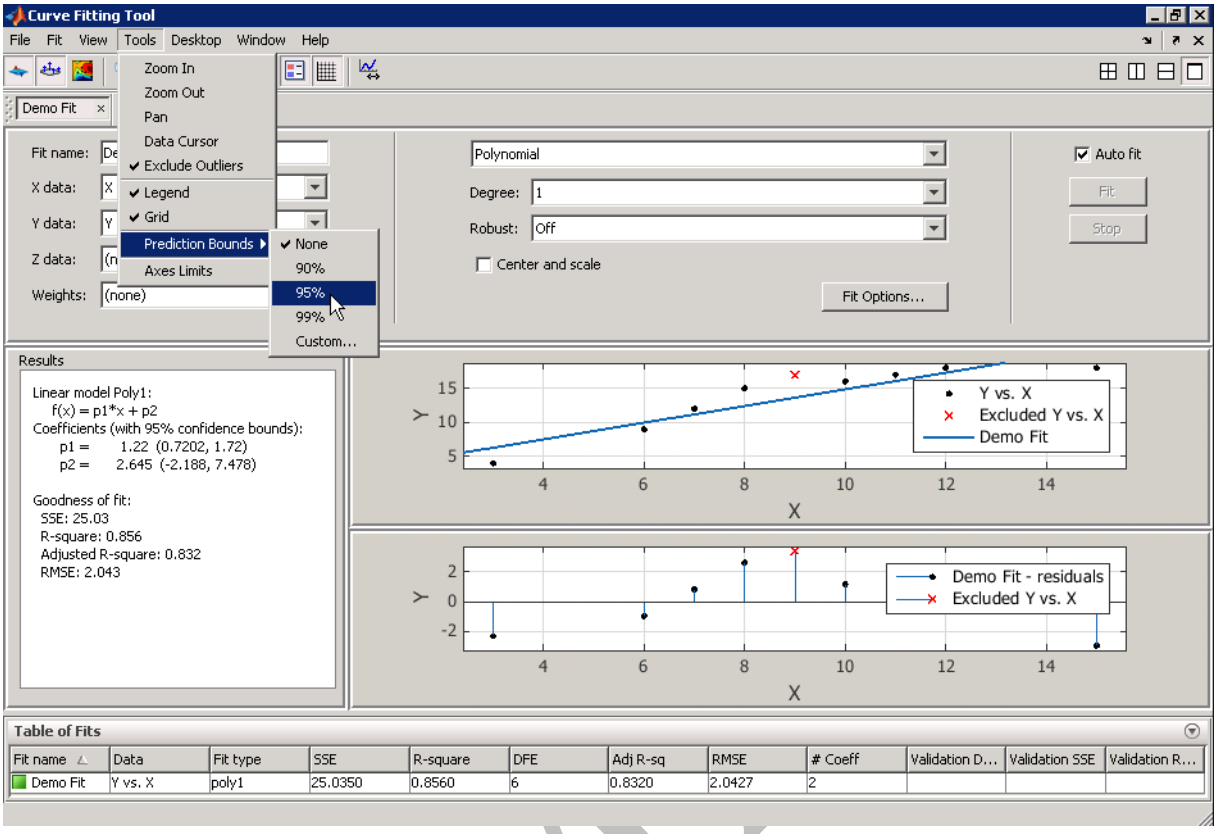


Once activated you have to click on outliers points to exclude them (not very scientific way...):

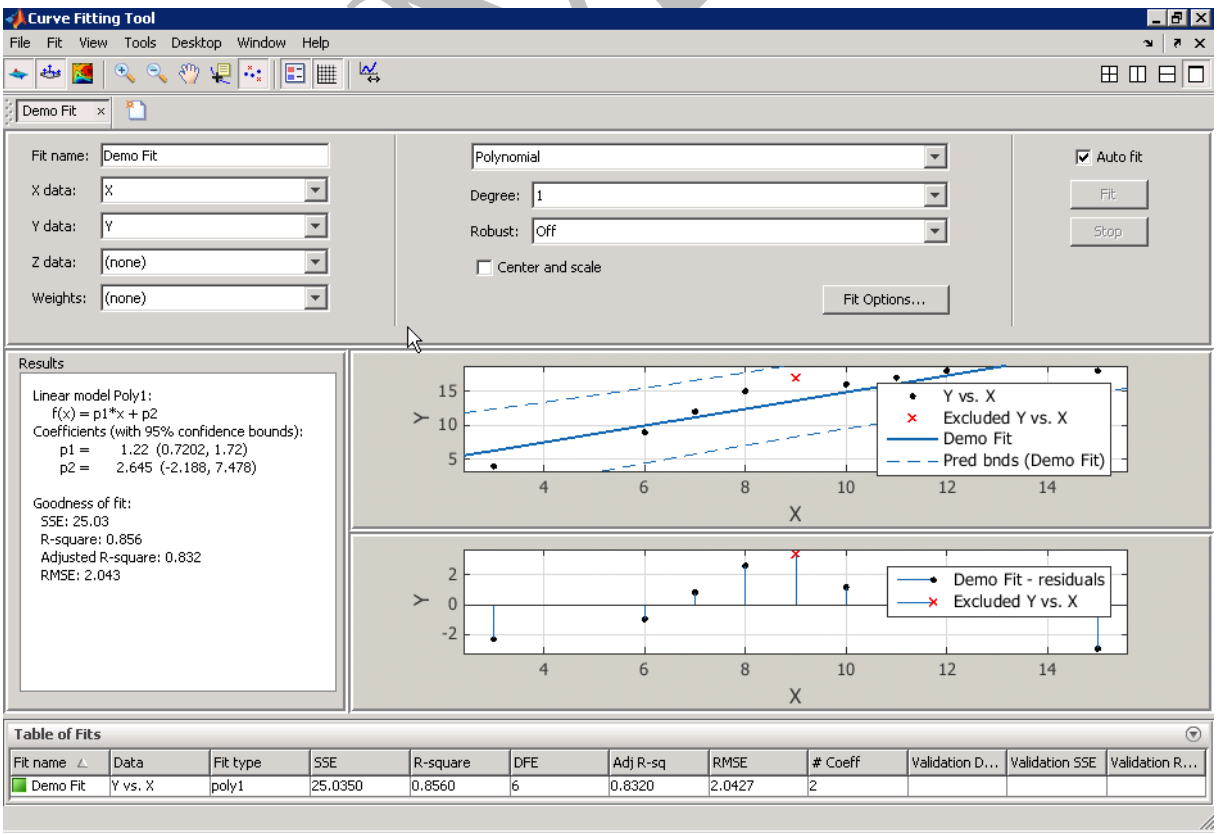


Thus the outlier elimination method here is a joke when we know the number of statistical technics dedicated to this purpose.

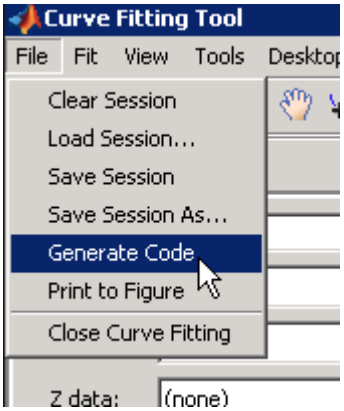
With this tool we also have the possibility to show the prediction bounds trough the menu **Tools/Prediction Bounds/95%**:



The result will be:



If you want to get the MATLAB script for all the previous manipulations you just have to go in the menu **File/Generate Code**:

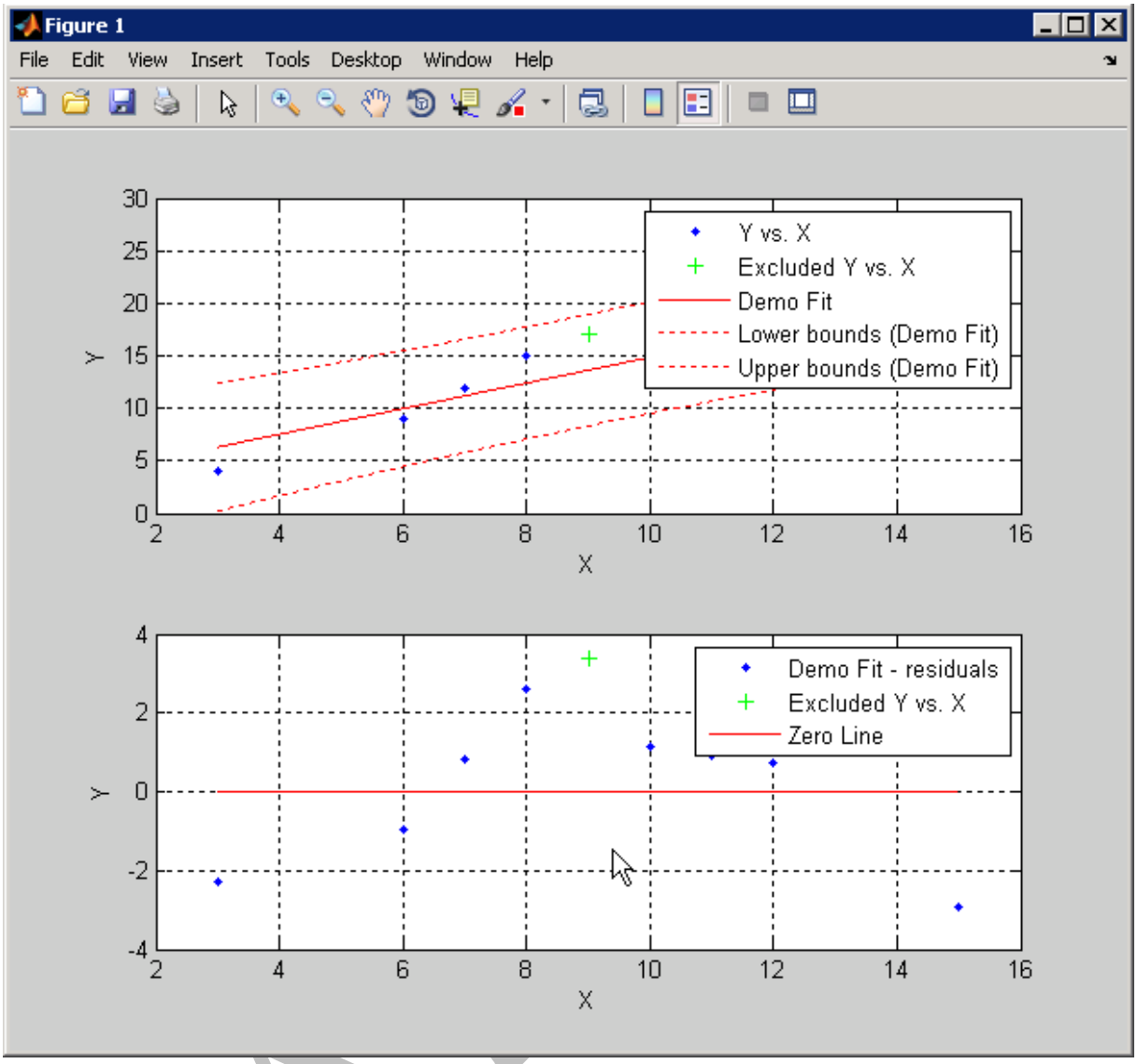


The result will be:

EDITOR PUBLISH VIEW

```
1   [xData, yData] = prepareCurveData( X, Y );
2
3   % Set up fittype and options.
4   ft = fittype( 'poly1' );
5   opts = fitoptions( ft );
6   opts.Lower = [-Inf -Inf];
7   opts.Upper = [Inf Inf];
8   ex = excludedata( xData, yData, 'Indices', 5 );
9   opts.Exclude = ex;
10
11  % Fit model to data.
12  [fitresult, gof] = fit( xData, yData, ft, opts );
13
14  % Plot fit with data.
15  subplot( 2, 1, 1 );
16  h = plot( fitresult, xData, yData, ex, 'predobs' );
17  legend( h, 'Y vs. X', 'Excluded Y vs. X', 'Demo Fit',...
18      'Lower bounds (Demo Fit)', 'Upper bounds (Demo Fit)',...
19      'Location', 'NorthEast' );
20  % Label axes
21  xlabel( 'X' );
22  ylabel( 'Y' );
23  grid on
24
25  % Plot residuals.
26  subplot( 2, 1, 2 );
27  h = plot( fitresult, xData, yData, ex, 'residuals' );
28  legend( h, 'Demo Fit - residuals', 'Excluded Y vs. X', 'Zero Line', 'Location', 'NorthEast' );
29  % Label axes
30  xlabel( 'X' );ylabel( 'Y' );grid on
```

And if you run it you will get:

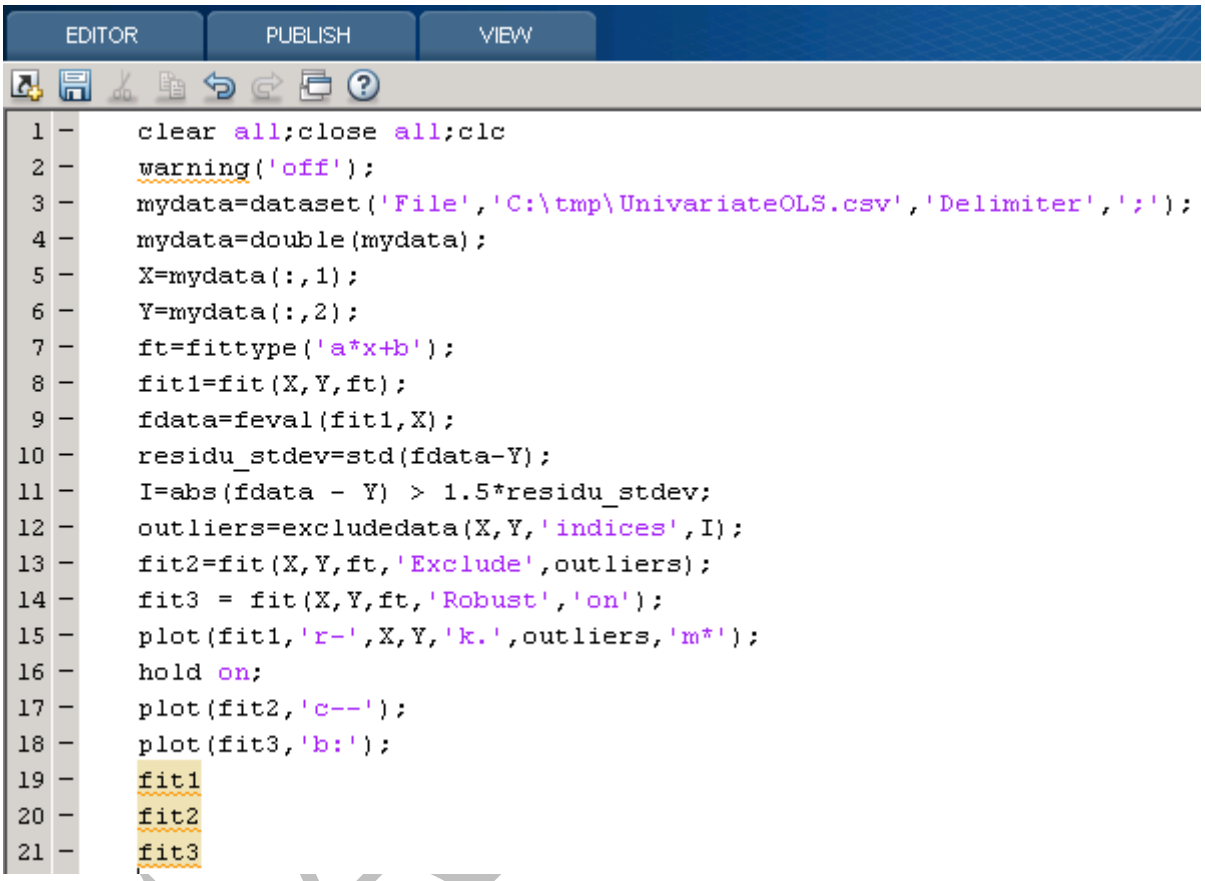

Figure 1
File Edit View Insert Tools Desktop Window Help
Y vs. X
Excluded Y vs. X
Demo Fit
Lower bounds (Demo Fit)
Upper bounds (Demo Fit)
Y
X
Demo Fit - residuals
Excluded Y vs. X
Zero Line
Y
X

SA

Manage outliers

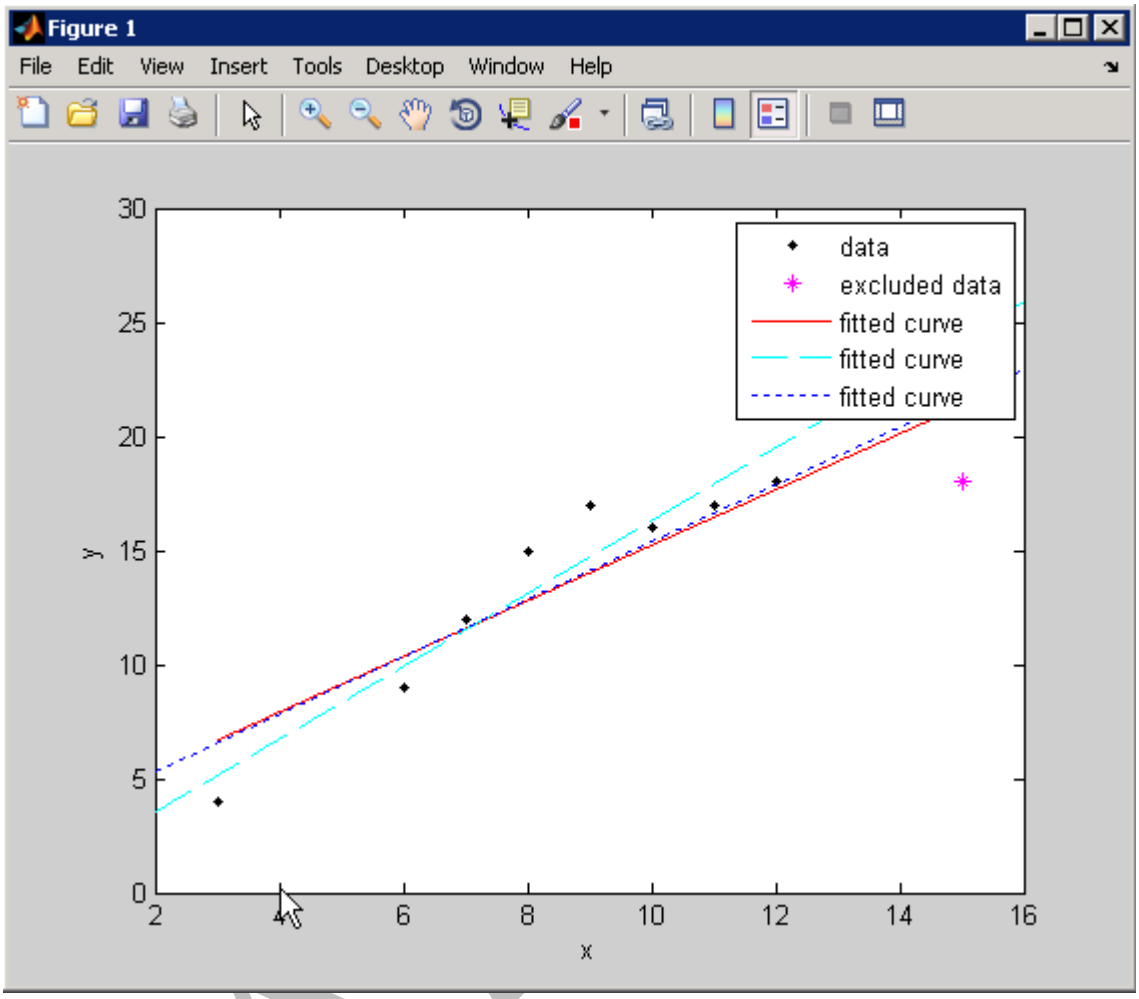
Manager outlier is a very important subject. Some trainings are given only on this subject during almost a whole day.

Here we show only a very basic example using the core function **feval( )** with the strong hypothesis of gaussian distributed residues to eliminate the outliers using the curve fitting toolbox functions **excludedata( )** and **fit( )** with various parameters:



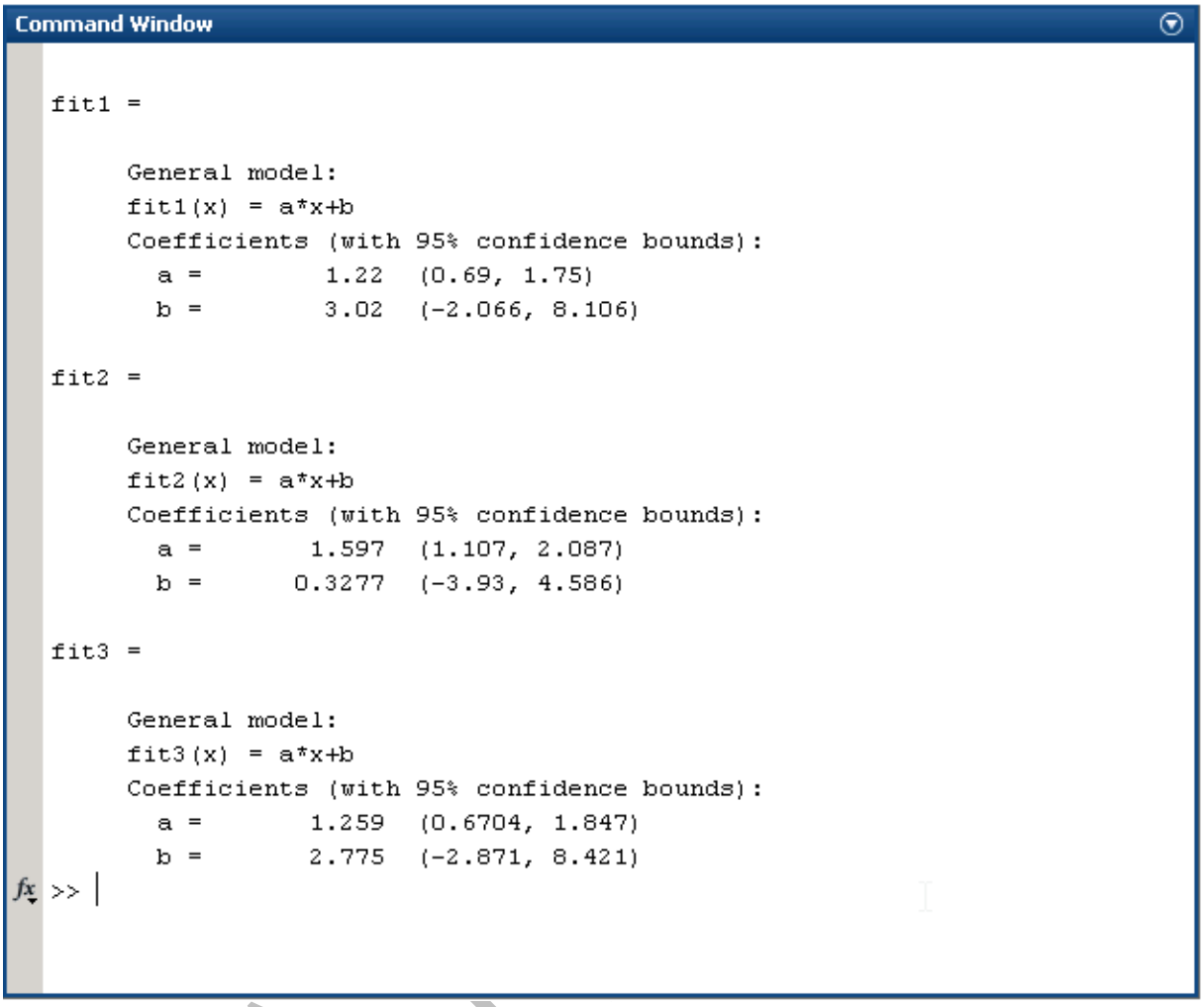

```
1 -    clear all;close all;clc
2 -    warning('off');
3 -    mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
4 -    mydata=double(mydata);
5 -    X=mydata(:,1);
6 -    Y=mydata(:,2);
7 -    ft=fittype('a*x+b');
8 -    fit1=fit(X,Y,ft);
9 -    fdata=feval(fit1,X);
10 -   residu_stdev=std(fdata-Y);
11 -   I=abs(fdata - Y) > 1.5*residu_stdev;
12 -   outliers=excludedata(X,Y,'indices',I);
13 -   fit2=fit(X,Y,ft,'Exclude',outliers);
14 -   fit3 = fit(X,Y,ft,'Robust','on');
15 -   plot(fit1,'r-',X,Y,'k.',outliers,'m*');
16 -   hold on;
17 -   plot(fit2,'c--');
18 -   plot(fit3,'b:');
19 -   fit1
20 -   fit2
21 -   fit3
```

So at the end we have a visual of three different regressions, in red the parametric regression including outliers, in dashed cyan the parametric regression without outliers and in dashed blue the non-parametric regression:



SA

And in the Command Window we have:



And if we need to extract some of the coefficients:

```
Command Window
>> fit1

fit1 =

     General model:
     fit1(x) = a*x+b
     Coefficients (with 95% confidence bounds):
       a =         1.22   (0.69, 1.75)
       b =         3.02   (-2.066, 8.106)
>> fit1.a

ans =

    1.2200

>> fit1.b

ans =

    3.0200

>> confint(fit1)

ans =

    0.6900   -2.0662
    1.7500    8.1062
```

SAMPLE

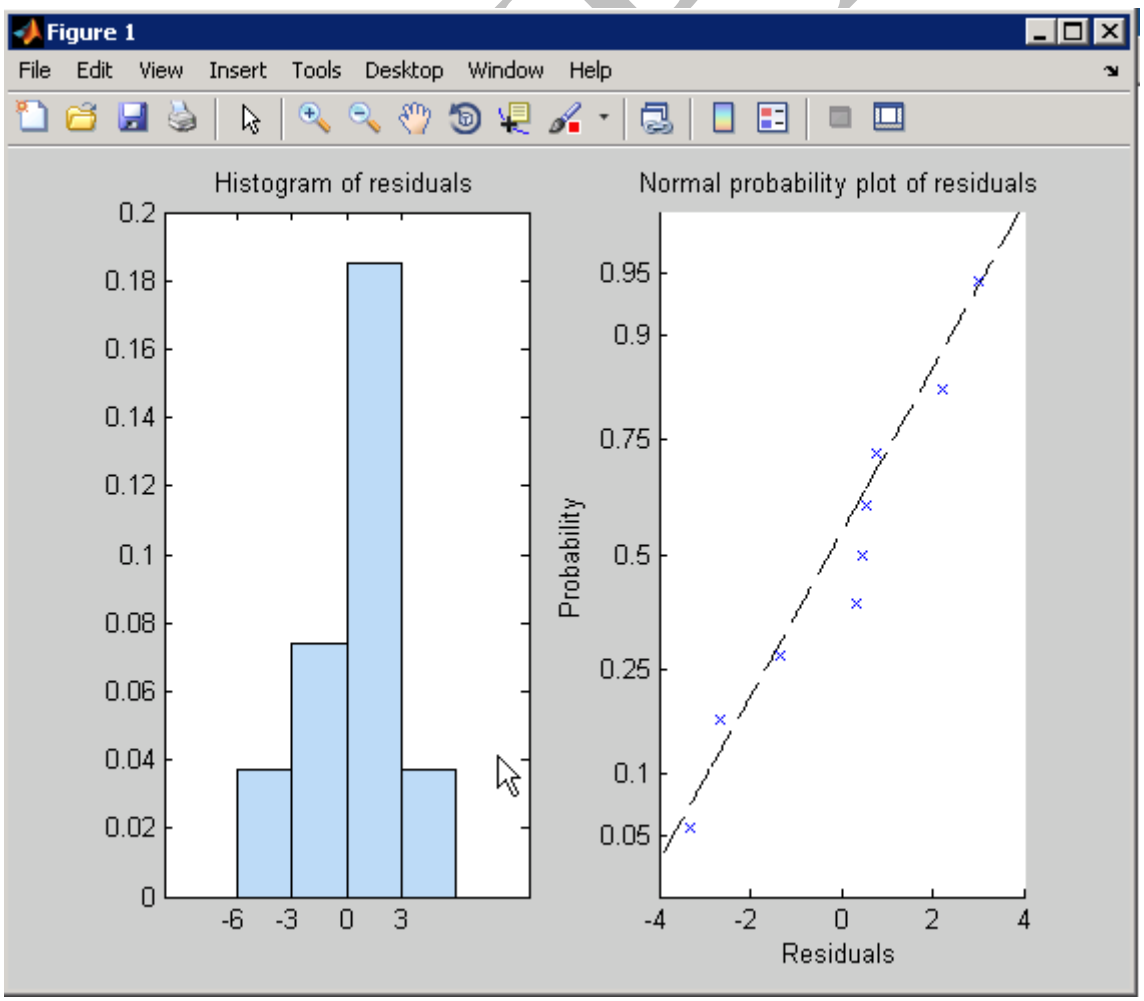
Residuals and Leverage (residual diagnosis)

We have proved in the theoretical course how to calculate the leverage of the residuals of a linear regression. Let us see if we obtain the same results as the one calculated by hand with our simple example.

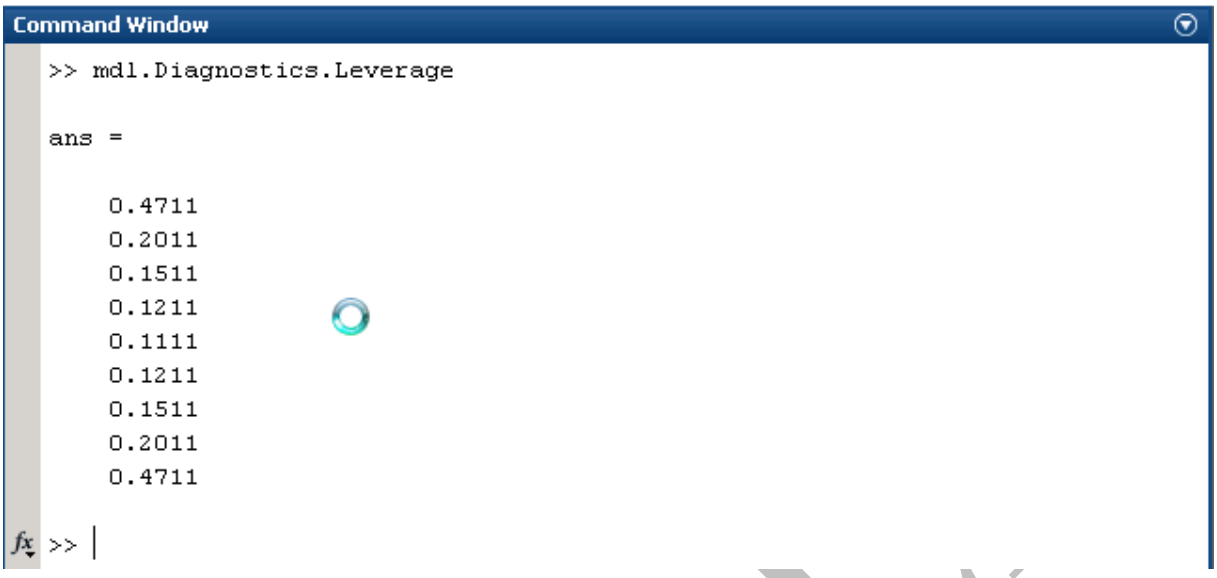
First we plot the residuals distribution and Henry plot using the function **plotResiduals( )**:

```
1   mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
2   mdl=LinearModel.fit(mydata,'linear');
3   subplot(1,2,1)
4   plotResiduals(mdl)
5   subplot(1,2,2)
6   plotResiduals(mdl,'probability')
7   mdl.Diagnostics.Leverage
```

So for the plots we get:



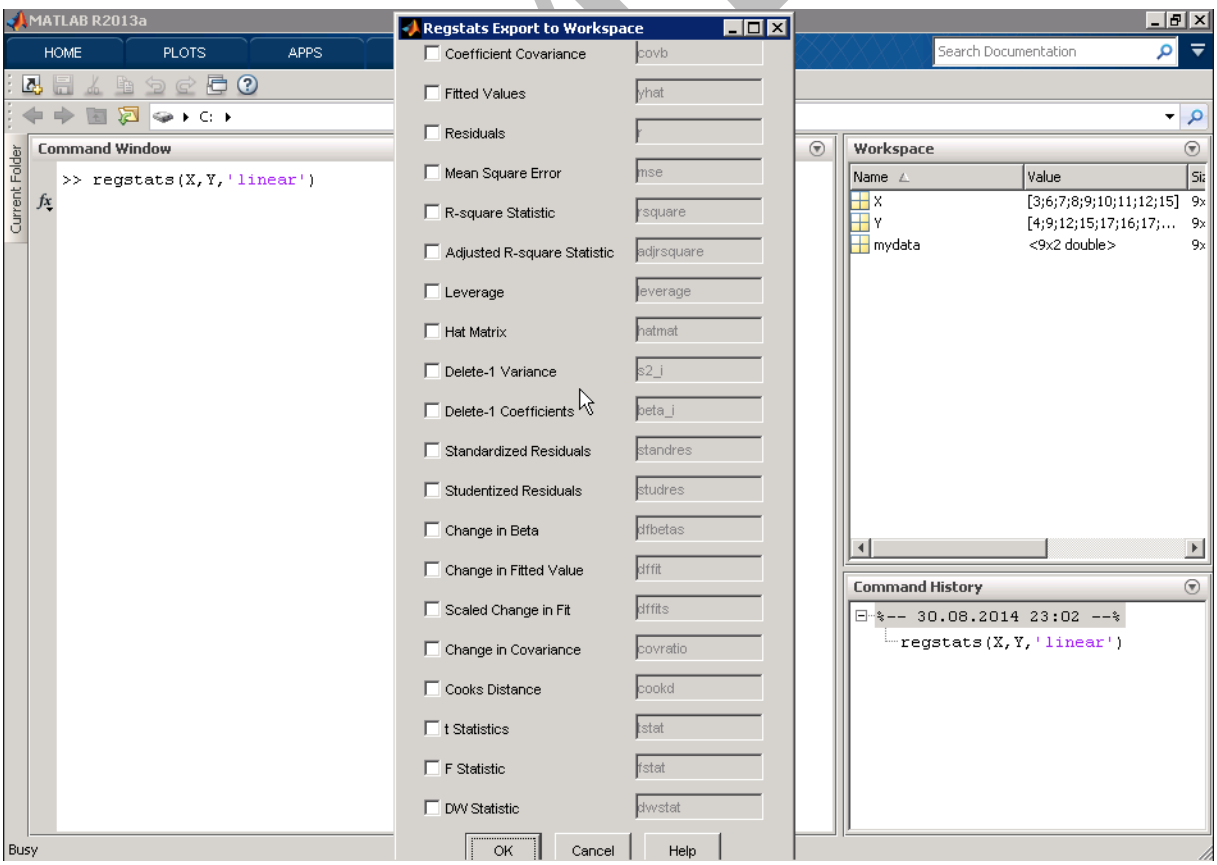
And for the diagnostics leverage we get using the function property Leverage of the method Diagnostics trough ....**Diagnostics.Leverage**:



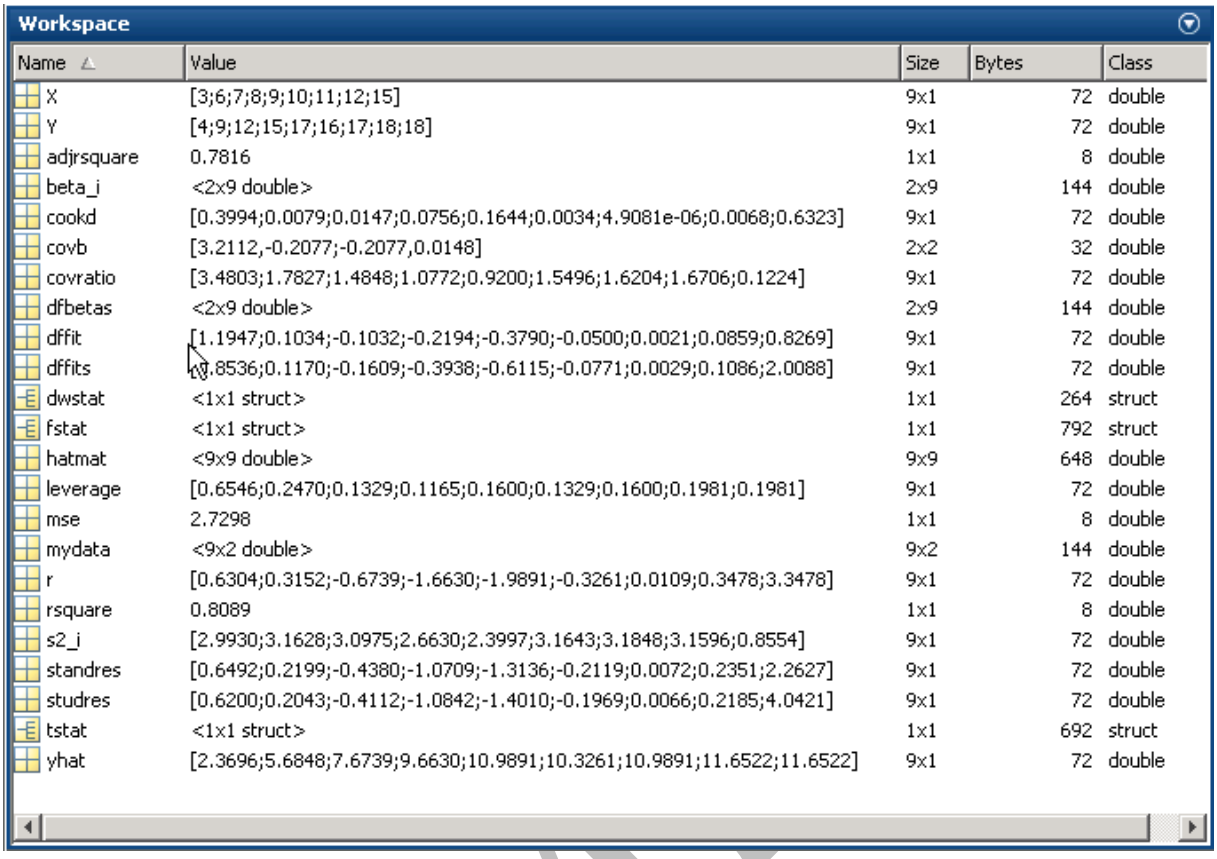
This is corresponding to what we get in the theoretical course.

Regression diagnostics wizard

We can get a lot of others diagnostics using the **regstats( )** function wizard:



If we select everything we get in the MATLAB variable workspace the following:



Workspace

Name	Value	Size	Bytes	Class
X	[3;6;7;8;9;10;11;12;15]	9x1	72	double
Y	[4;9;12;15;17;16;17;18;18]	9x1	72	double
adjrsquare	0.7816	1x1	8	double
beta_i	<2x9 double>	2x9	144	double
cookd	[0.3994;0.0079;0.0147;0.0756;0.1644;0.0034;4.9081e-06;0.0068;0.6323]	9x1	72	double
covb	[3.2112,-0.2077;-0.2077,0.0148]	2x2	32	double
covratio	[3.4803;1.7827;1.4848;1.0772;0.9200;1.5496;1.6204;1.6706;0.1224]	9x1	72	double
dfbetas	<2x9 double>	2x9	144	double
dffit	[1.1947;0.1034;-0.1032;-0.2194;-0.3790;-0.0500;0.0021;0.0859;0.8269]	9x1	72	double
dffits	[0.8536;0.1170;-0.1609;-0.3938;-0.6115;-0.0771;0.0029;0.1086;2.0088]	9x1	72	double
dwstat	<1x1 struct>	1x1	264	struct
fstat	<1x1 struct>	1x1	792	struct
hatmat	<9x9 double>	9x9	648	double
leverage	[0.6546;0.2470;0.1329;0.1165;0.1600;0.1329;0.1600;0.1981;0.1981]	9x1	72	double
mse	2.7298	1x1	8	double
mydata	<9x2 double>	9x2	144	double
r	[0.6304;0.3152;-0.6739;-1.6630;-1.9891;-0.3261;0.0109;0.3478;3.3478]	9x1	72	double
rsquare	0.8089	1x1	8	double
s2_i	[2.9930;3.1628;3.0975;2.6630;2.3997;3.1643;3.1848;3.1596;0.8554]	9x1	72	double
standres	[0.6492;0.2199;-0.4380;-1.0709;-1.3136;-0.2119;0.0072;0.2351;2.2627]	9x1	72	double
studres	[0.6200;0.2043;-0.4112;-1.0842;-1.4010;-0.1969;0.0066;0.2185;4.0421]	9x1	72	double
tstat	<1x1 struct>	1x1	692	struct
yhat	[2.3696;5.6848;7.6739;9.6630;10.9891;10.3261;10.9891;11.6522;11.6522]	9x1	72	double

SAMPLE

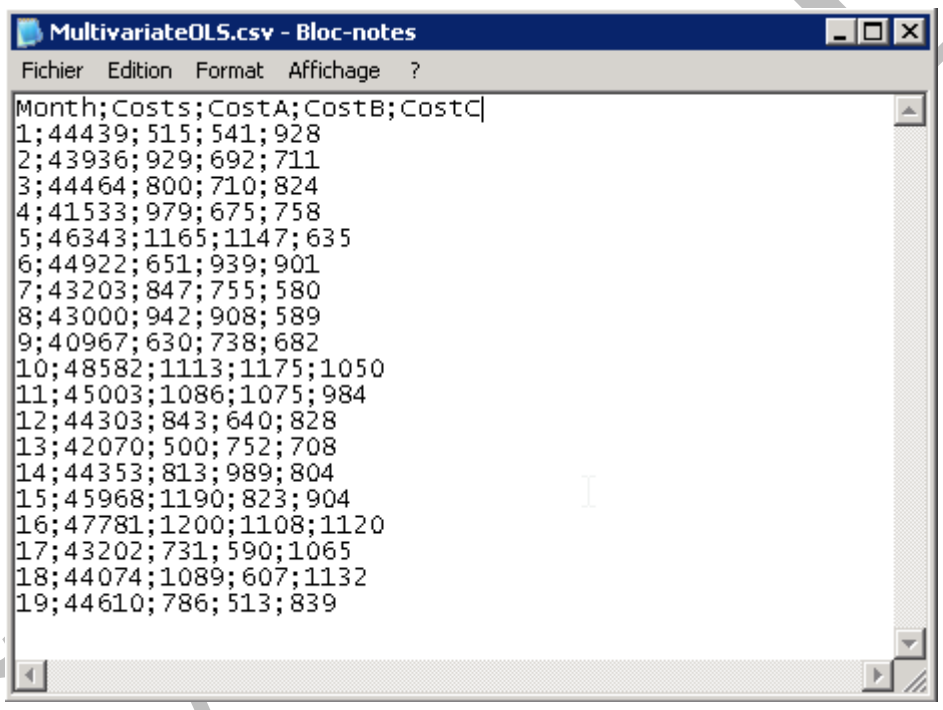
Ordinary Least Squares (parametric) multivariate regression (with/without interactions)

Ok now that we have finished our small overview of the univariate parametric linear regression techniques we saw during the theoretical course we will now focus on some of the multivariate techniques we have studied in details.

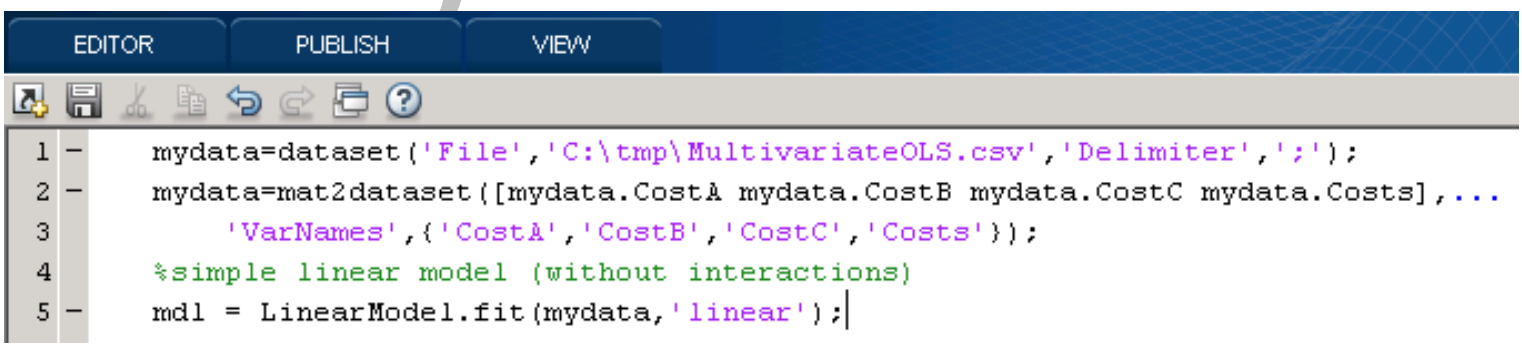
We will for the MATLAB examples, as always until now, use the same data as the one used in the theoretical course.

1st method (coefficient and constant cannot be extracted)

We begin first with the tests the student-t test for slopes and constants using the following datas:



To obtain the coefficient and student t-test the first method use the function **LinearModel.fit( )**:



We get then the following:

```
Command Window

mdl =


Linear regression model:
    Costs ~ 1 + CostA + CostB + CostC

Estimated Coefficients:
                   Estimate     SE        tStat      pValue
    (Intercept)     35103       1837.2    19.106     6.112e-12
    CostA            2.066       1.665    1.2408       0.23373
    CostB           4.1764      1.6813    2.4841      0.025288
    CostC           4.7906      1.7893    2.6774      0.017223


Number of observations: 19, Error degrees of freedom: 15
Root Mean Squared Error: 1.25e+03
R-squared: 0.645,  Adjusted R-Squared 0.575
F-statistic vs. constant model: 9.1, p-value = 0.00113
fx >>
```

And to obtain the ANOVA of the regression we just use the function **anova( )** as we already saw it for the univariate regression:

```
Command Window

>> anova(mdl)

ans =

              SumSq          DF     MeanSq          F          pValue
    CostA     2.4164e+06      1     2.4164e+06     1.5397      0.23373
    CostB     9.6843e+06      1     9.6843e+06     6.1706     0.025288
    CostC      1.125e+07      1      1.125e+07     7.1682     0.017223
    Error     2.3541e+07     15     1.5694e+06

fx >>
```

As for the univariate regression then we can explicitly write the model we are looking for:

```
Command Window
>> %and now with interactions (quadratic)
>> mdl=LinearModel.fit(mydata,'Costs~1+CostA+CostB+CostC+CostA*CostB')

mdl =


Linear regression model:
    Costs ~ 1 + CostC + CostA*CostB

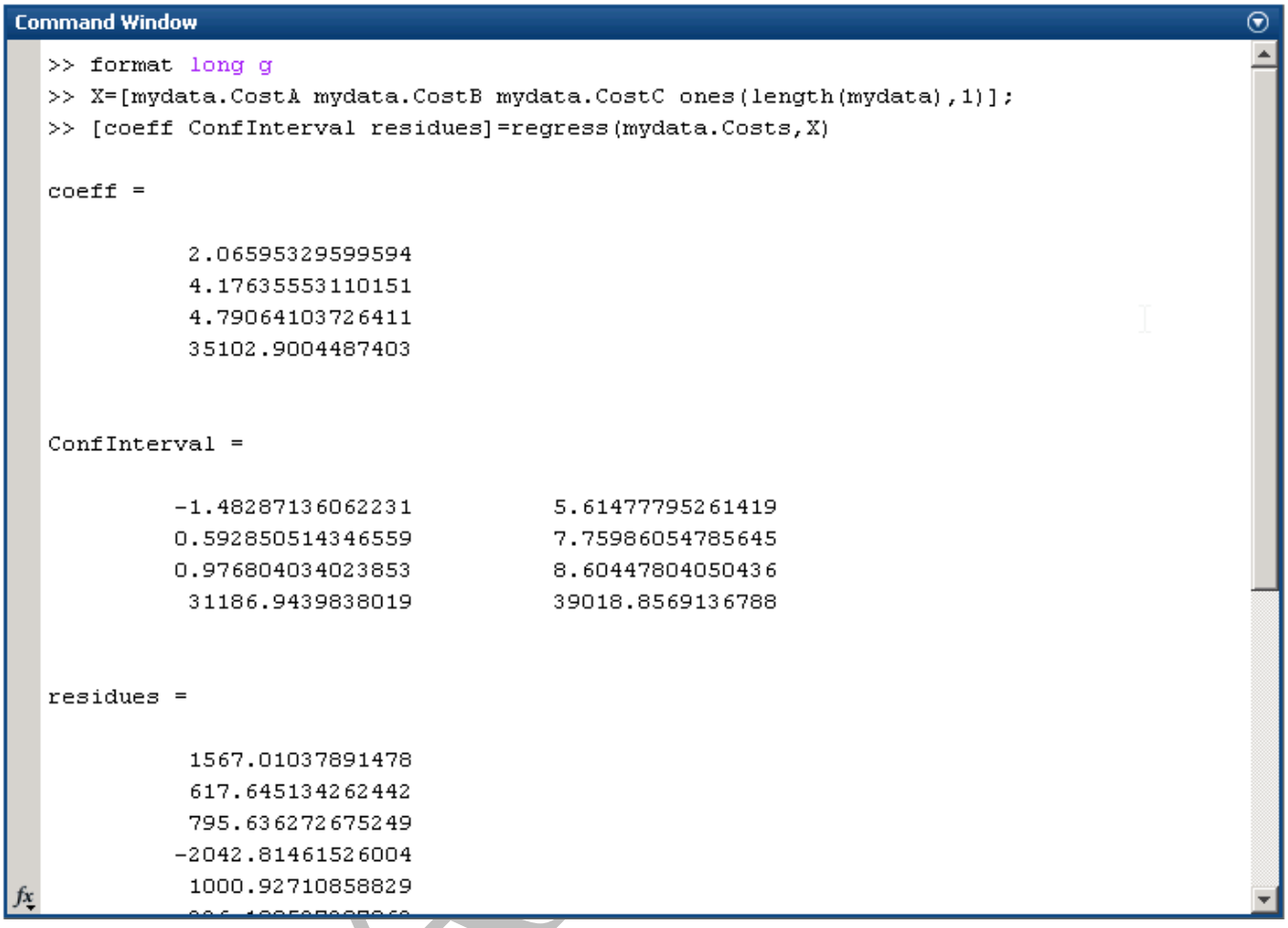
Estimated Coefficients:
                   Estimate     SE           tStat      pValue
    (Intercept)      45507        5909.1      7.7012    2.1269e-06
    CostA           -8.2599       5.8238     -1.4183       0.17798
    CostB           -8.7341       7.1917     -1.2145       0.24466
    CostC            4.0788       1.7067      2.3899       0.03147
    CostA:CostB     0.01322     0.0071883      1.8391      0.087205


Number of observations: 19, Error degrees of freedom: 14
Root Mean Squared Error: 1.16e+03
R-squared: 0.714,  Adjusted R-Squared 0.633
F-statistic vs. constant model: 8.76, p-value = 0.000929
fx >>
```

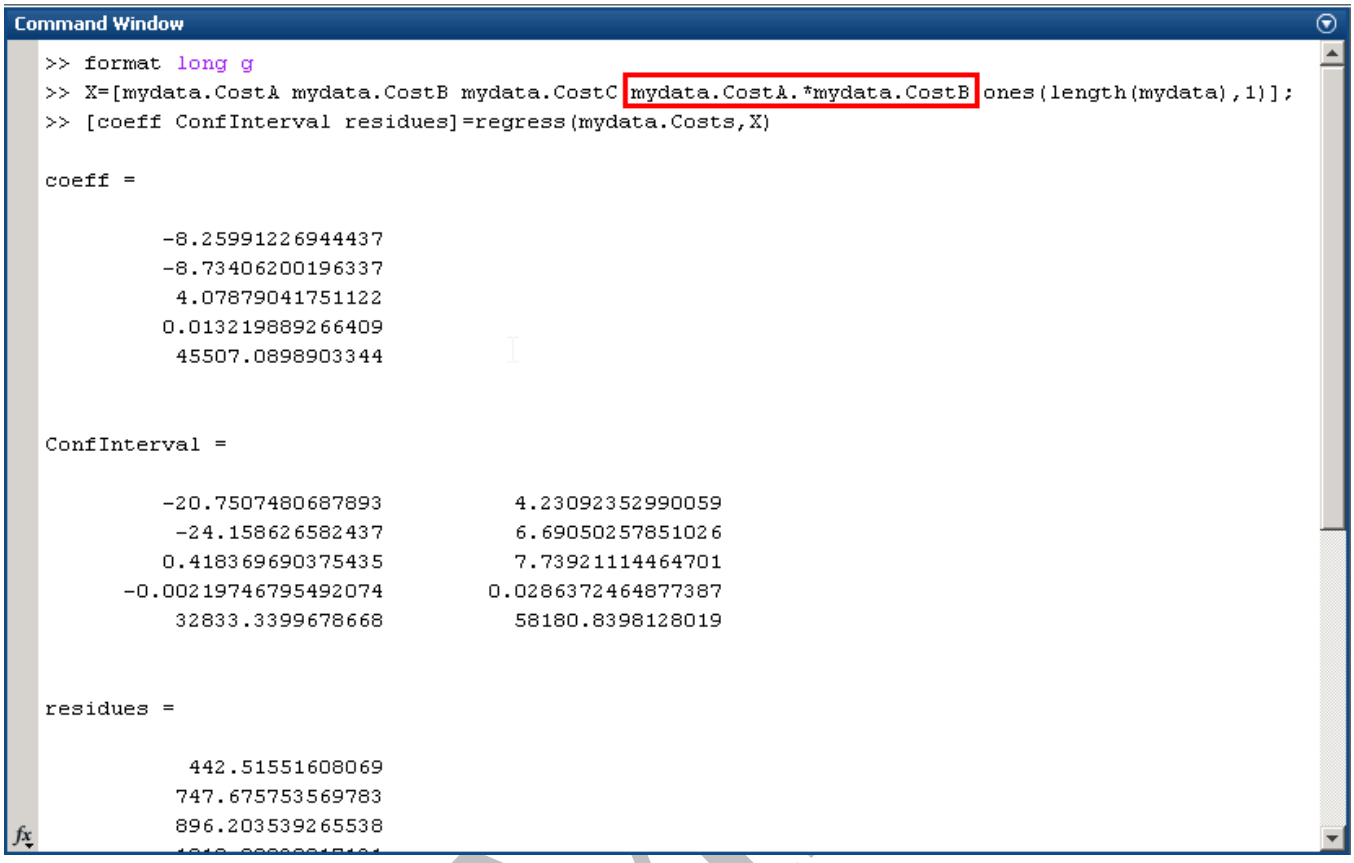
SAMPLE

2nd method (coefficient and constant and others can be extracted)

Now have a look to the second method that a in my point of view more useful because we can access directly to the coefficients, confidence interval and residues of the model. This second method use also as for the univariate regression the function **regress( )**:



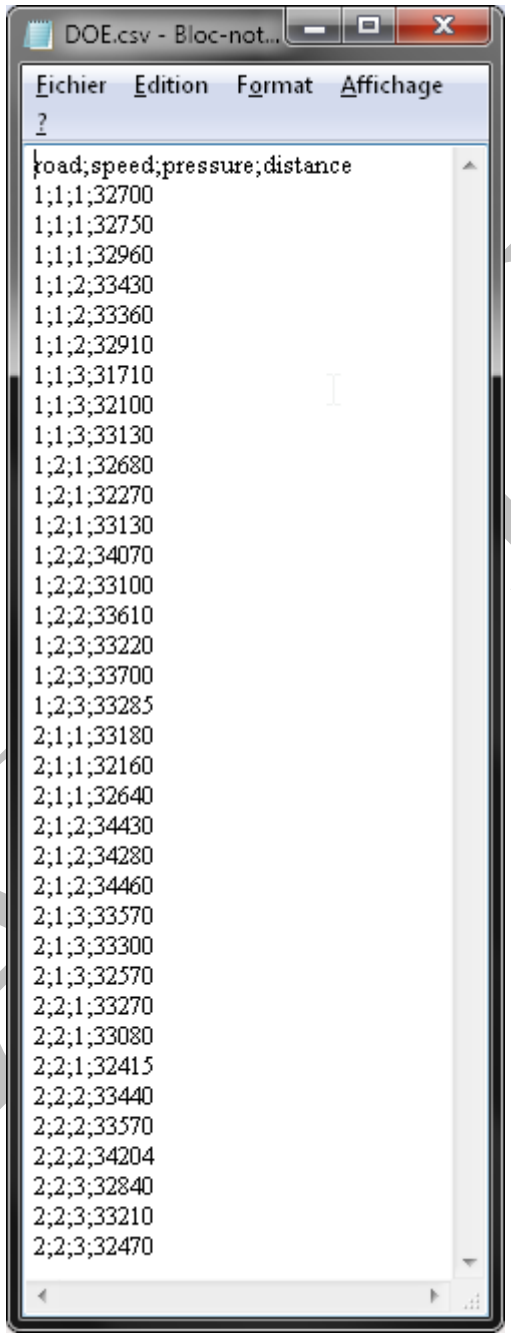
If we want to change the model by adding an interaction we just have to write the input matrix as following:



SAMPLE

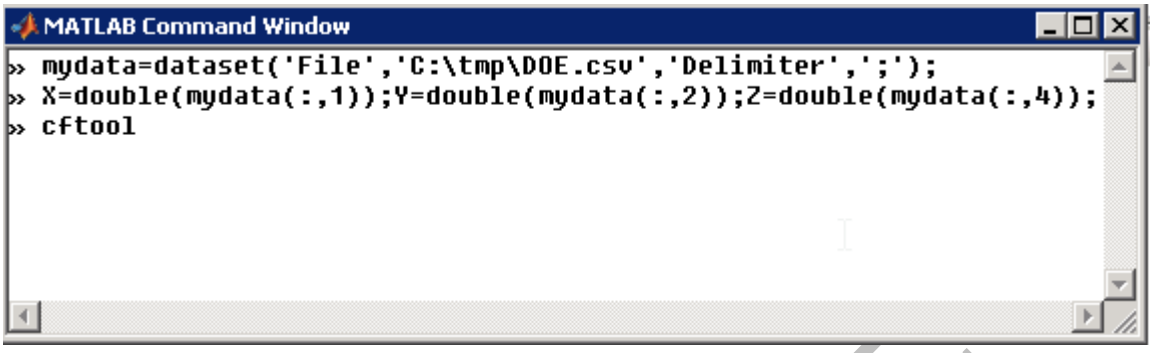
Polynomial multivariate regression using OLS and Curve fit Toolbox

We will see here that the curvefit tool is limited to 2 explanatory variables... Consider the following dataset (the same as in the theoretical course):

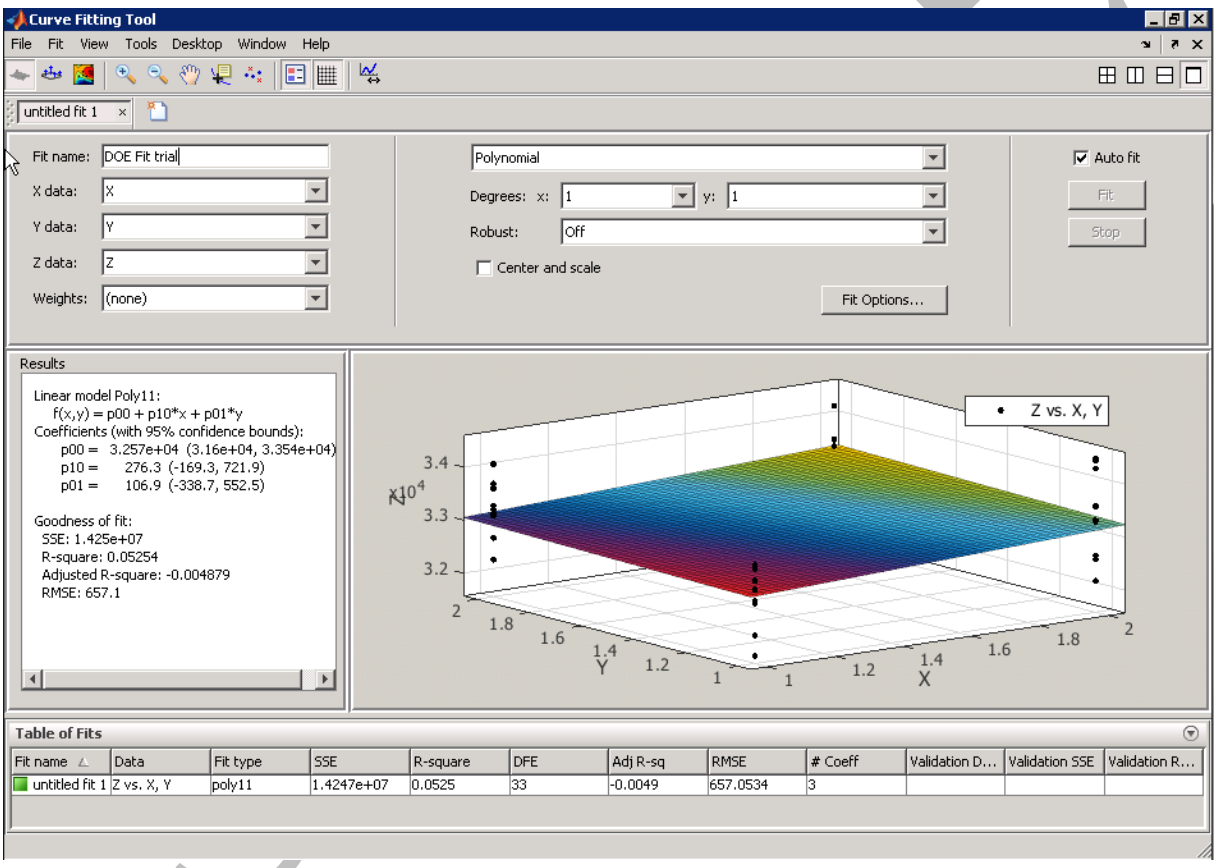


SAMPLE

We import the data using the same technic as always and we launch the curve fitting toolbox by typing **cftool**:



The following appears:



As you can see the tool is limited to two explanatory variables and also without interaction. In my point of view this makes the tool almost useless but perhaps there is an unknown tip to me to manager more than two explanatory variables.

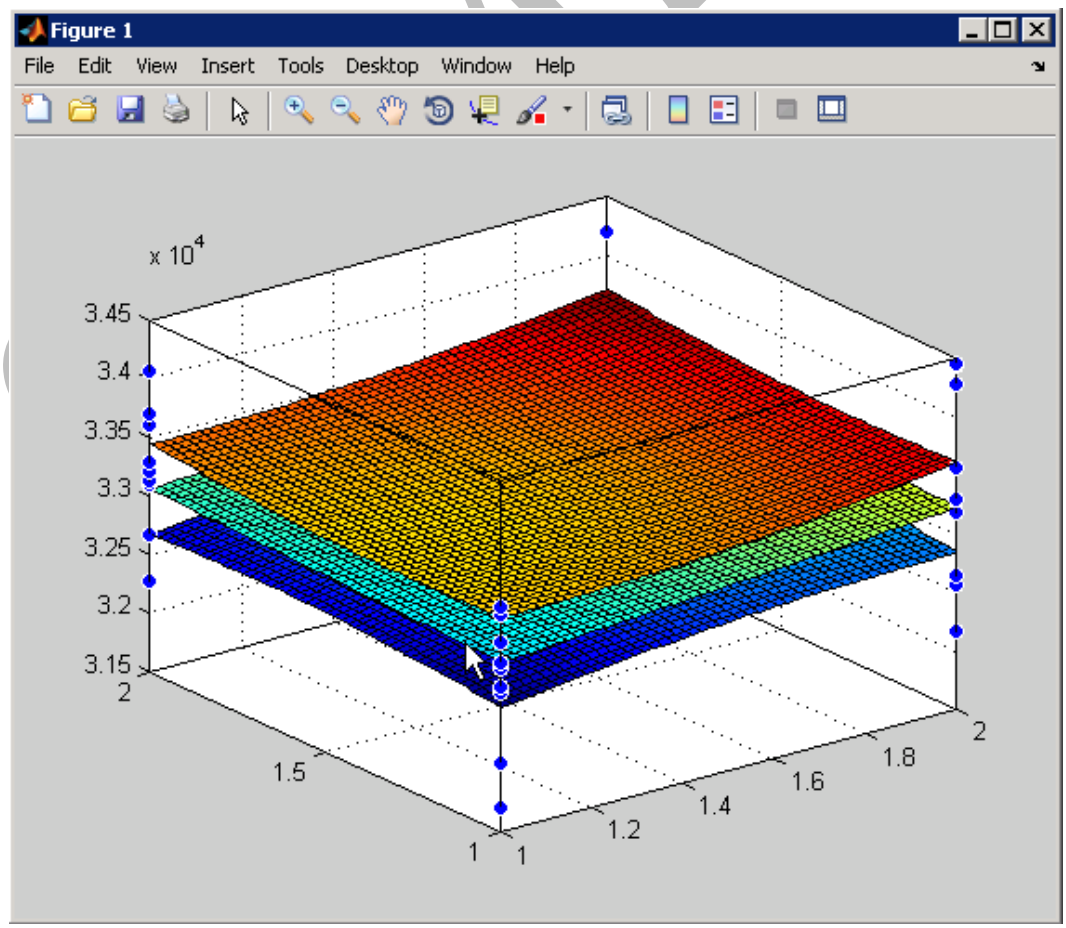
We can otherwise the same stuff by using the function **fit( )** and by indicating we want a first order polynomial on *x* and *y* this is why we write the argument *poly11*:

```
MATLAB Command Window
>> mydata=dataset('File','C:\tmp\DOE.csv','Delimiter',';');
>> X=double(mydata(:,1));Y=double(mydata(:,2));Z=double(mydata(:,4));
>> surffit = fit([X,Y],Z,'poly11')

     Linear model Poly11:
     surffit(x,y) = p00 + p10*x + p01*y
     Coefficients (with 95% confidence bounds):
       p00 =   3.257e+04  (3.16e+04, 3.354e+04)
       p10 =       276.3  (-169.3, 721.9)
       p01 =       106.9  (-338.7, 552.5)

>> plot(surffit,[X,Y],Z,'Style','predfunc')
>>
```

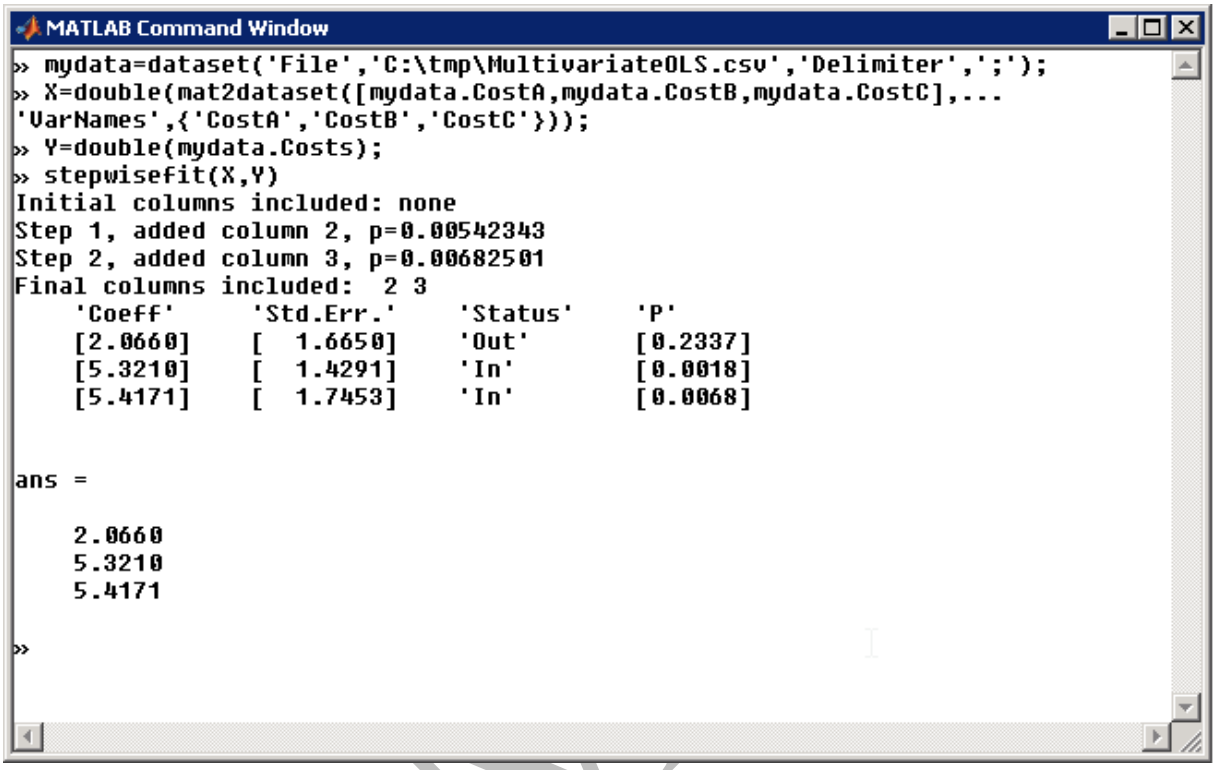
We the get the coefficients, the confidence intervals and also the plot of the function itself:



(Forward) Stepwise regression

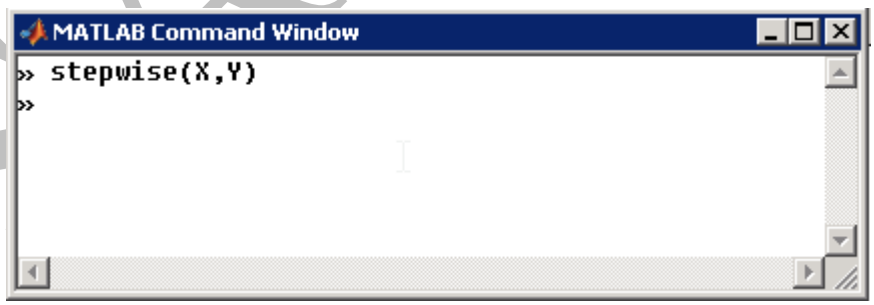
Stepwise regression is a must know method to choose the most accurate explanatory variable and to not get to much degrees of freedom that biased the R squares.

For this purpose we use the **stepwisefit( )** function:

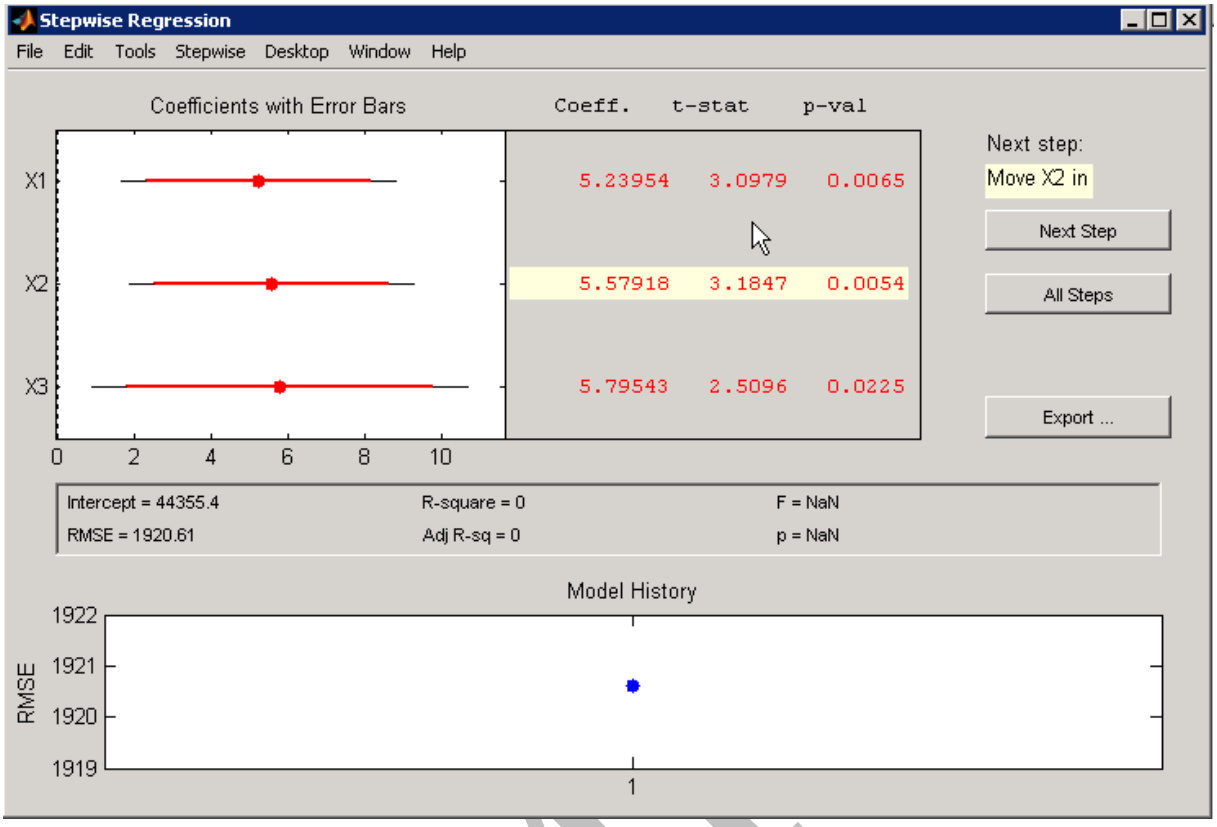


As we can see it we get the same results as in the theoretical course.

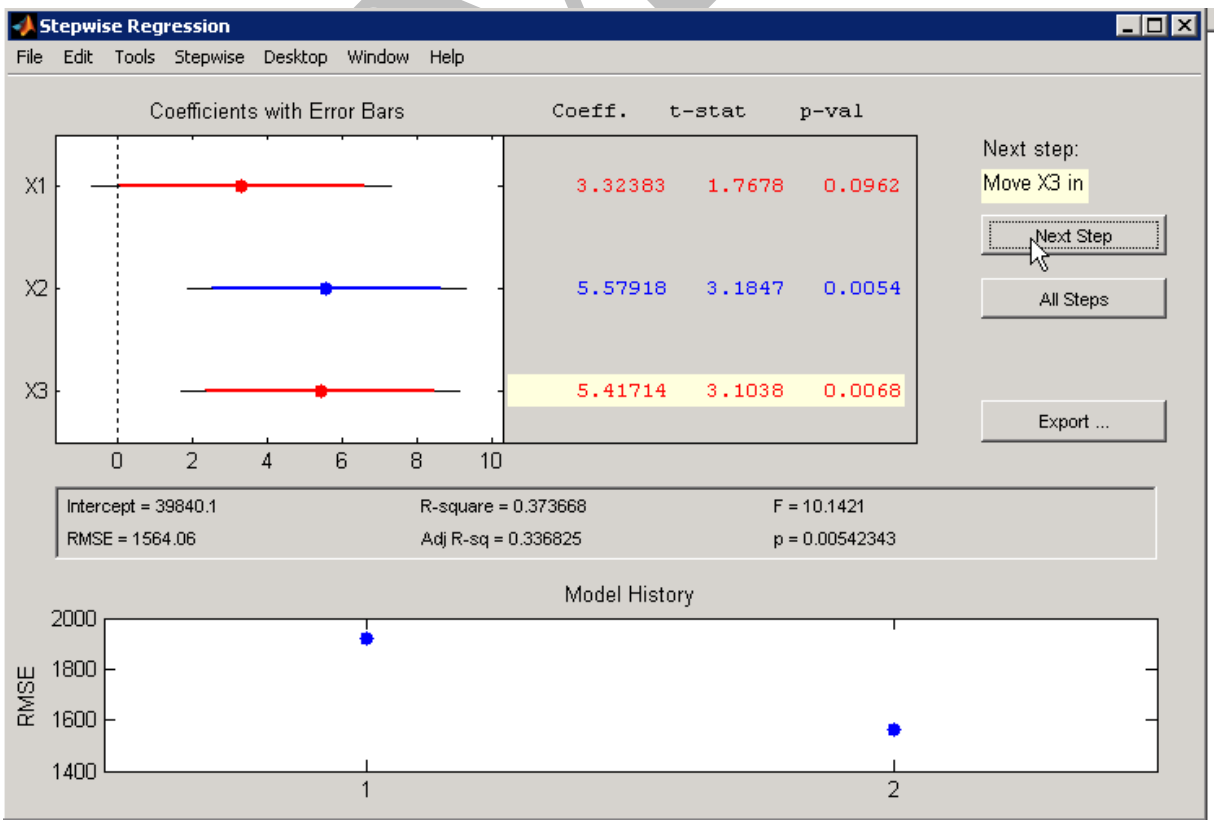
Now if instead of **stepwisefit( )** function we use **stepwise( )** function:



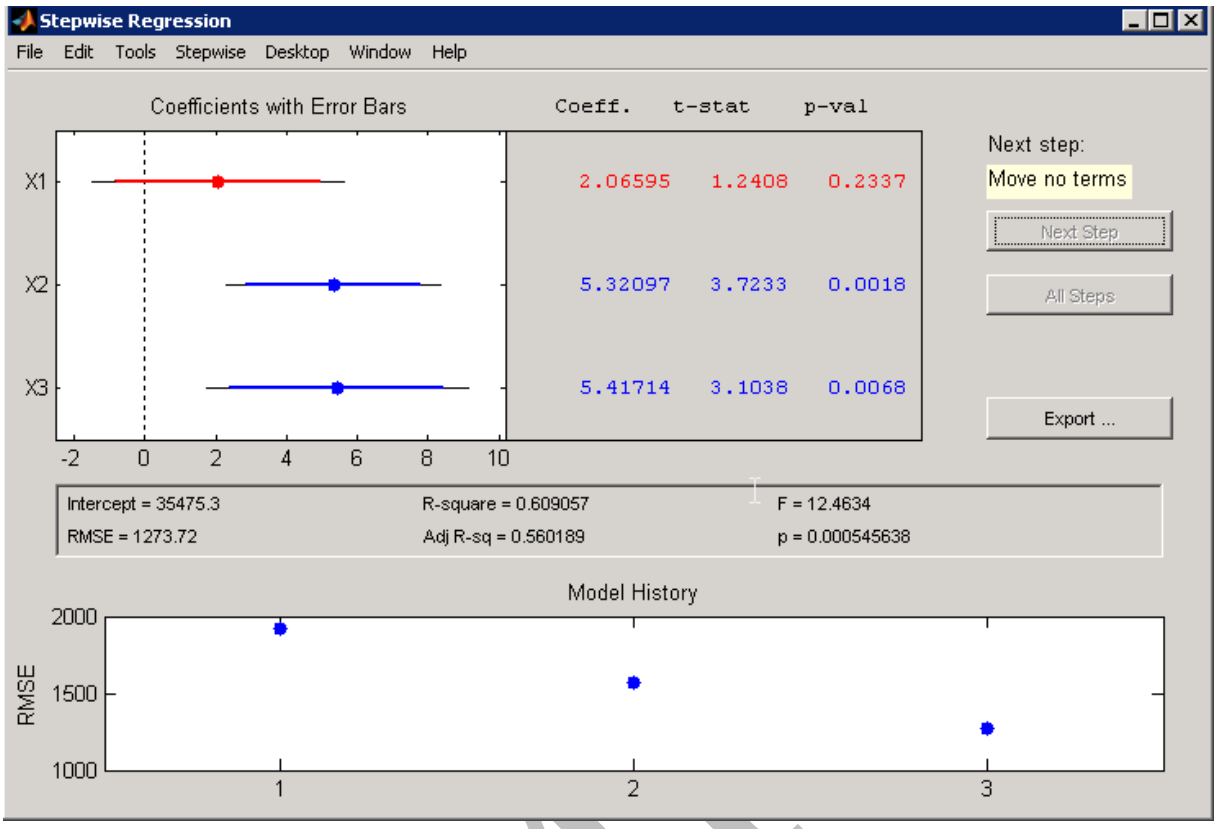
We get a figure that resume the situation step by step with a wizard. This is the first step:



This is the second step when you click on **Next step**:



And this is the last one:



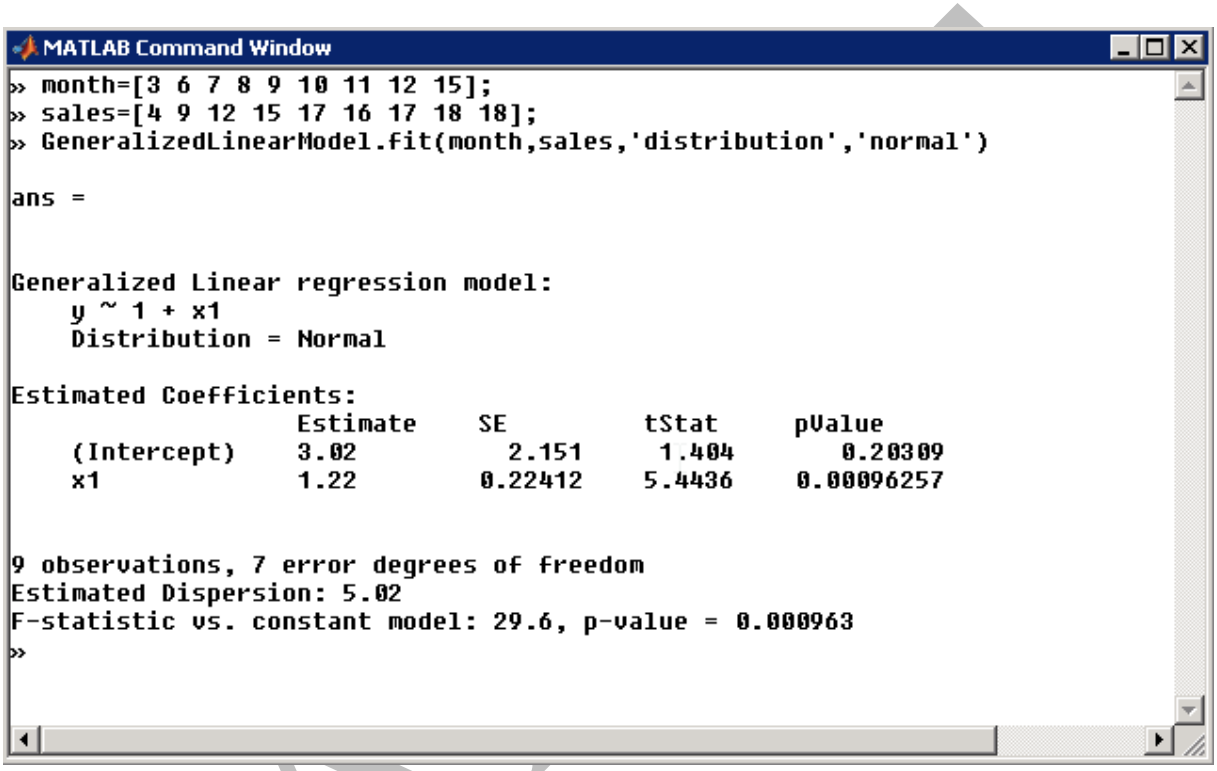
SAMPLE

General Linear Models (GLM)

Ok this is a very big subject that has needed a lot of time in the theoretical course. We haven't proved all the post-hoc tests of all these regression models but we saw what was the idea behind them.

Normal regression

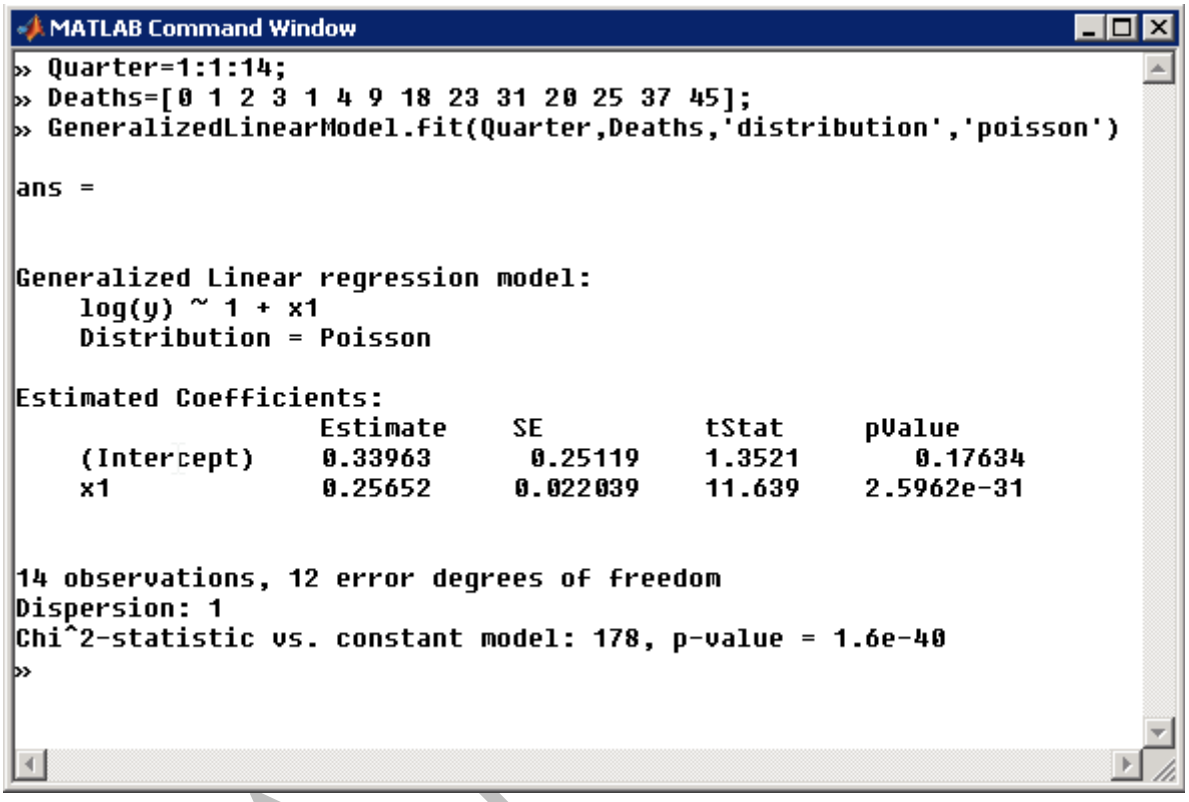
Using the function **GeneralizedLinearModel.fit( )** consider first the Normal parametric regression model:



The results are as expected.

Log-linear Poisson regression

The log-linear Poisson regression model was not easy to prove in the theoretical course. We saw that we have to take especially care of model assumption because Poisson mean is equal to its variance. But a part of this, let us take a look if we obtain the same results as the calculation made by hand using the same data and the function **GeneralizedLinearModel.fit( )**:



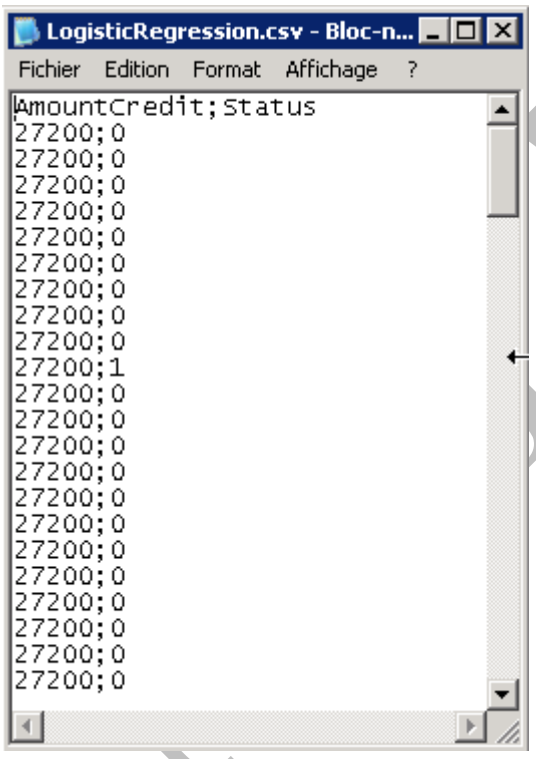
As we see it we obtain the same results!

SAM

Binomial (logistic) regression

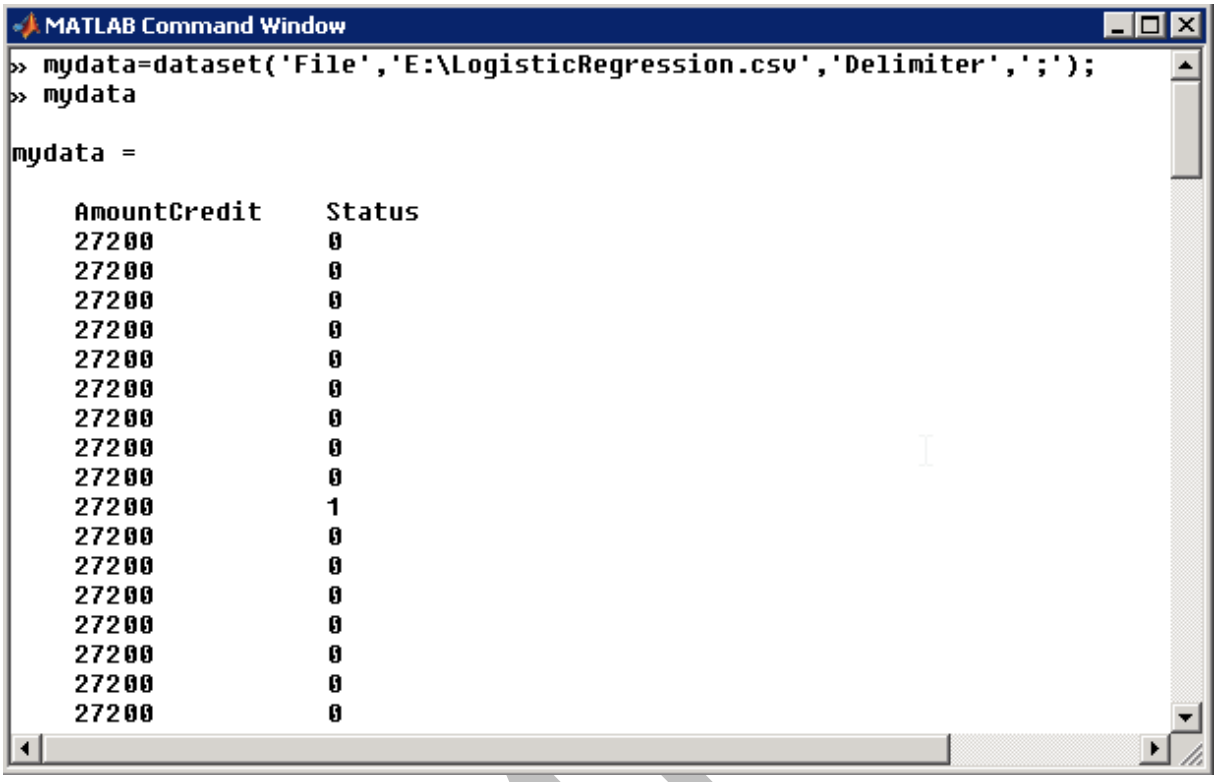
The binomial logistic regression is the second most important GLM, used a lot in Credit Risk or Social Sciences we have discussed and make a lot of proofs about this model in the theoretical course. We did not prove all the corresponding confidence interval tests but the major results also this is why we can still see how to use it in MATLAB as we did it for R and Minitab.

Consider the same datas:

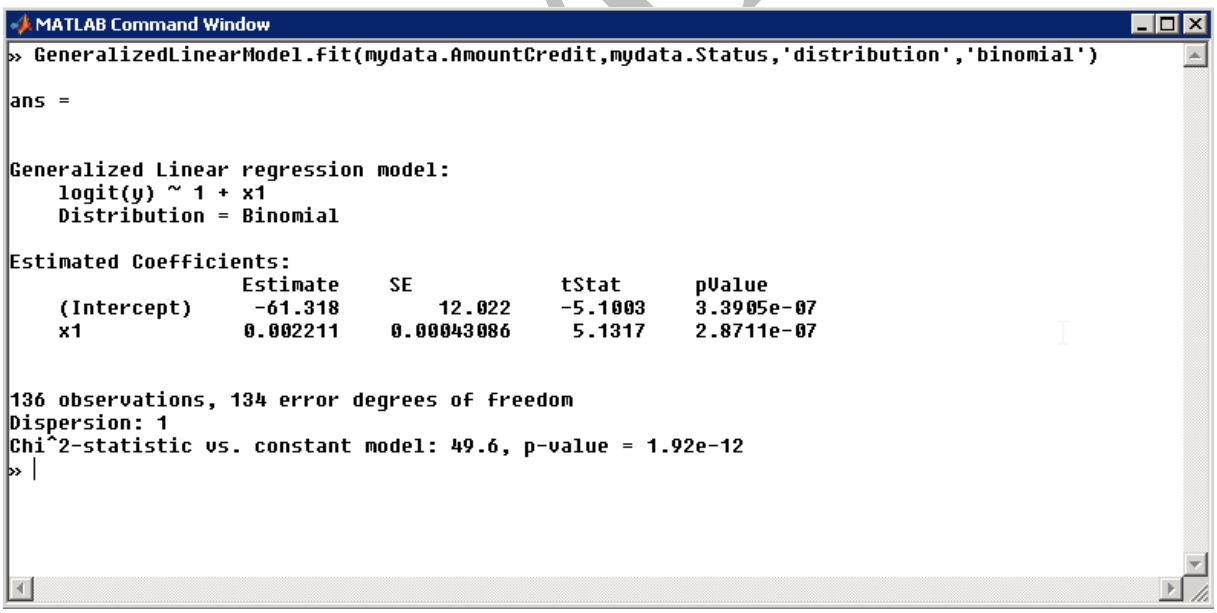


SAMPLE

First we load the data as we always do:

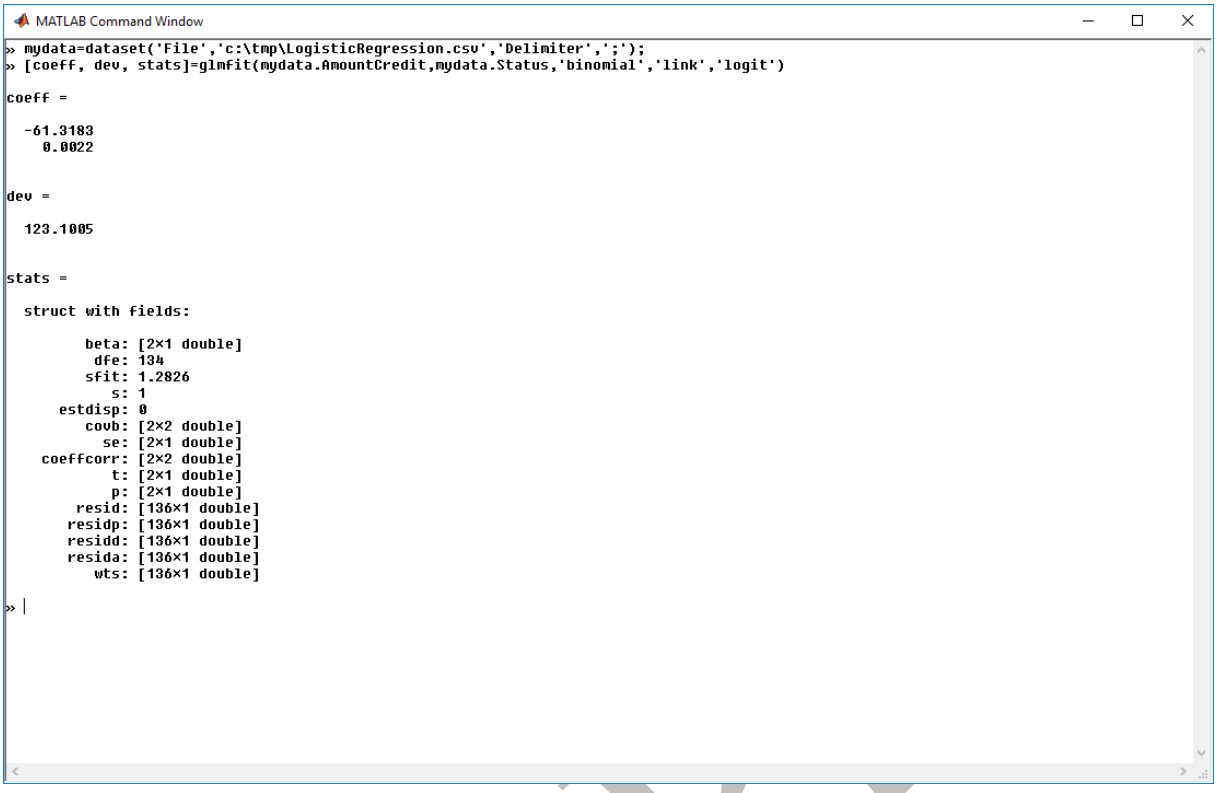


And we regress using once again the function **GeneralizedLinearModel.fit( )**:



And the estimate and the coefficient correspond to what we get by hand and in the other softwares.

Since at least MATLAB 2017a the syntax has change quite a bit. We have to use the function **glmfit( )**:


MATLAB Command Window
» mydata=dataset('File','c:\tmp\LogisticRegression.csv','Delimiter',';');
» [coeff, dev, stats]=glmfit(mydata.AmountCredit,mydata.Status,'binomial','link','logit')
coeff =
-61.3183
0.0022
dev =
123.1005
stats =
struct with fields:
beta: [2×1 double]
dfe: 134
sfit: 1.2826
s: 1
estdisp: 0
covb: [2×2 double]
se: [2×1 double]
coeffcorr: [2×2 double]
t: [2×1 double]
p: [2×1 double]
resid: [136×1 double]
residp: [136×1 double]
residd: [136×1 double]
resida: [136×1 double]
wts: [136×1 double]
»

SAMPLE

Canonical Correlation

We will use the same data set as in the theoretical course:

CanonicalCorrelation - Bloc-notes

Fichier Edition Format Affichage ?

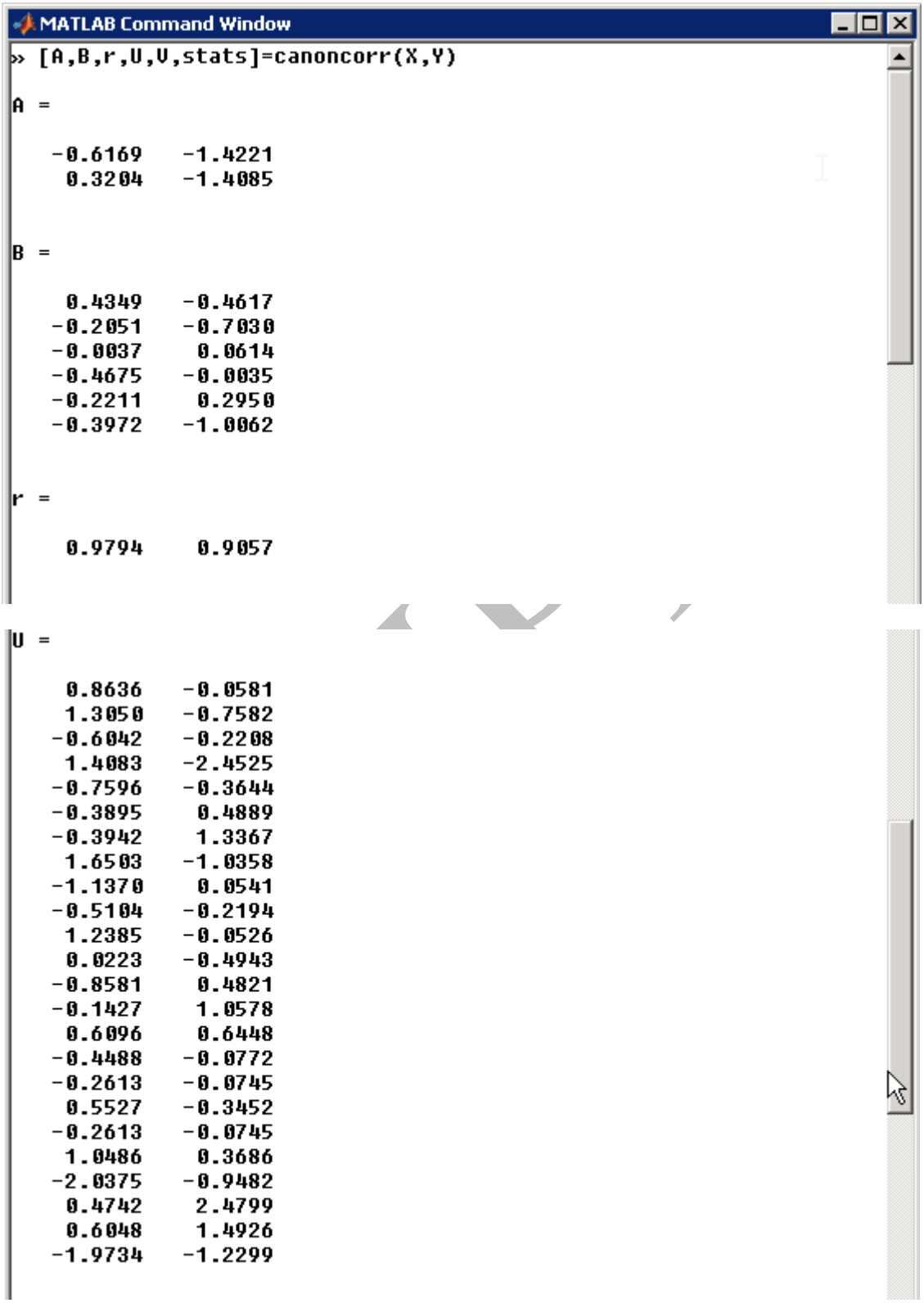
```
Type;Model;Economy;Service;Value;Price;Design;Sport.;Safety;Easy h.
Audi;100;3.9;2.8;2.2;4.2;3;3.1;2.4;2.8
BMW;5 series;4.8;1.6;1.9;5;2;2.5;1.6;2.8
Citroen;AX;3;3.8;3.8;2.7;4;4.4;4;2.6
Ferrari;;5.3;2.9;2.2;5.9;1.7;1.1;3.3;4.3
Fiat;Uno;2.1;3.9;4;2.6;4.5;4.4;4.4;2.2
Ford;Fiesta;2.3;3.1;3.4;2.6;3.2;3.3;3.6;2.8
Hyundai;;2.5;3.4;3.2;2.2;3.3;3.3;3.3;2.4
Jaguar;;4.6;2.4;1.6;5.5;1.3;1.6;2.8;3.6
Lada;Samara;3.2;3.9;4.3;2;4.3;4.5;4.7;2.9
Mazda;323;2.6;3.3;3.7;2.8;3.7;3;3.7;3.1
Mercedes;200;4.1;1.7;1.8;4.6;2.4;3.2;1.4;2.4
Mitsubishi;Galant;3.2;2.9;3.2;3.5;3.1;3.1;2.9;2.6
Nissan;Sunny;2.6;3.3;3.9;2.1;3.5;3.9;3.8;2.4
Opel;Corsa;2.2;2.4;3;2.6;3.2;4;2.9;2.4
Opel;Vectra;3.1;2.6;2.3;3.6;2.8;2.9;2.4;2.4
Peugeot;306;2.9;3.5;3.6;2.8;3.2;3.8;3.2;2.6
Renault;19;2.7;3.3;3.4;3;3.1;3.4;3;2.7
Rover;;3.9;2.8;2.6;4;2.6;3;3.2;3
Toyota;Corolla;2.5;2.9;3.4;3;3.2;3.1;3.2;2.8
Volvo;;3.8;2.3;1.9;4.2;3.1;3.6;1.6;2.4
Trabant;601;3.6;4.7;5.5;1.5;4.1;5.8;5.9;3.1
Vw;Golf;2.4;2.1;2;2.6;3.2;3.1;3.1;1.6
Vw;Passat;3.1;2.2;2.1;3.2;3.5;3.5;2.8;1.8
Wartburg;1.3;3.7;4.7;5.5;1.7;4.8;5.2;5.5;4
```

We first prepare the data:

MATLAB Command Window

```
» mydata=dataset('file','c:\tmp\CanonicalCorrelation.csv','Delimiter',';');
Warning: Variable names were modified to make them valid MATLAB identifiers.
> In genvalidnames at 81
  In @dataset\private\setvarnames at 40
  In dataset.readFile at 237
  In dataset.dataset>dataset.dataset at 351
» X=double(mydata(:,[5,6]));
» Y=double(mydata(:,[3:4,7:10]));
»
```

Now we use the function **canoncorr( )** to get:



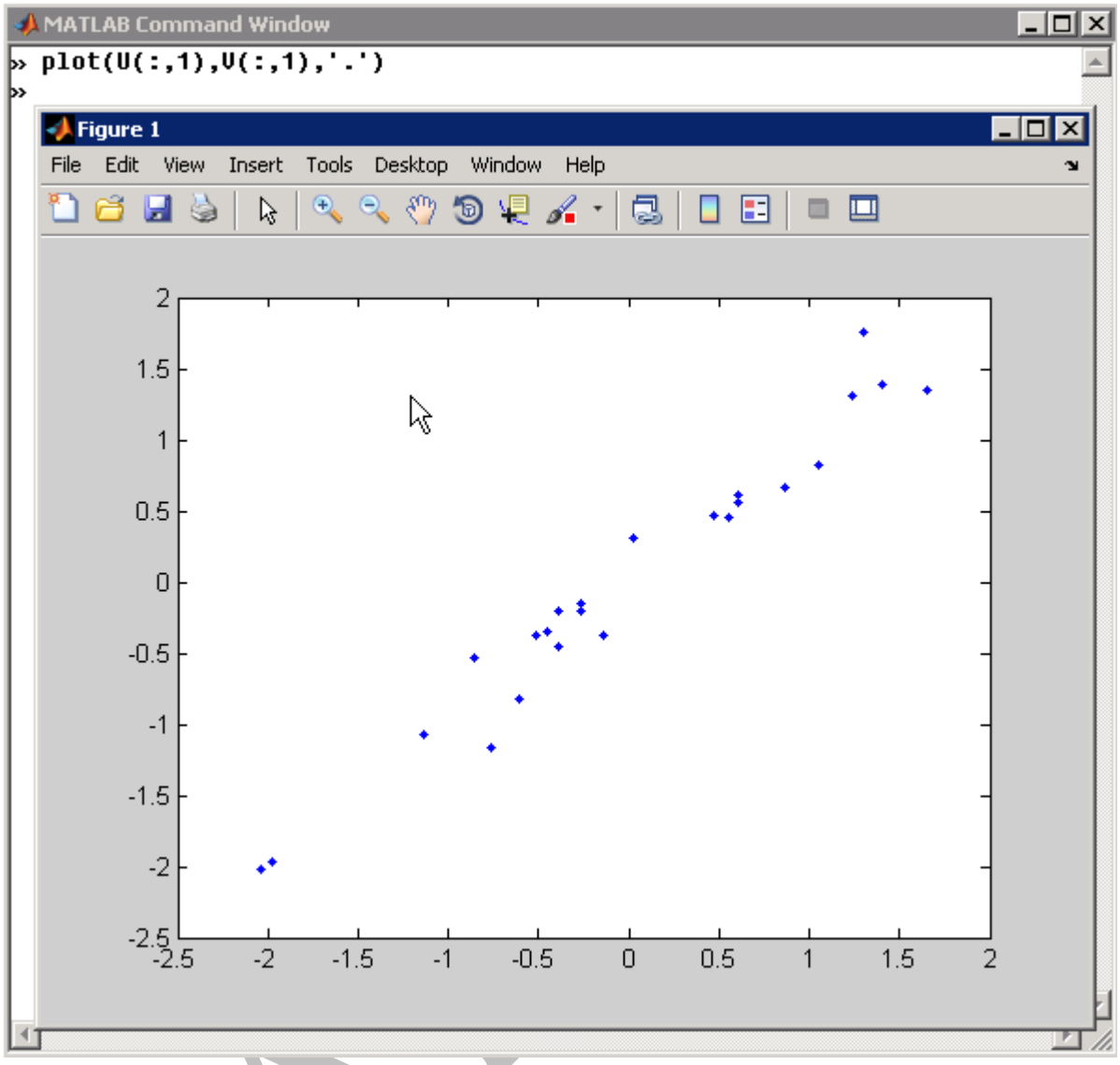
```
V =

    0.6601   -0.4763
    1.7587   -0.3436
   -0.8222   -0.0335
    1.3935   -2.5096
   -1.1655    0.8629
   -0.4568    0.4172
   -0.2065    0.4340
    1.3479   -1.3044
   -1.0775   -0.2734
   -0.3702   -0.1025
    1.3081    0.2749
    0.3037    0.1315
   -0.5343    0.6159
   -0.3703    1.1489
    0.6063    0.4245
   -0.3438   -0.0595
   -0.1979    0.0090
    0.4520   -0.4657
   -0.1469    0.3481
    0.8207    0.0922
   -2.0194   -0.8844
    0.4724    2.1346
    0.5551    1.4684
   -1.9670   -1.9091
```

```
stats =

     Wilks: [0.0073 0.1798]
       df1: [12 5]
       df2: [32 17]
         F: [28.4743 15.5112]
        pF: [1.1178e-13 8.1432e-06]
     chisq: [90.9347 31.8185]
    pChisq: [3.2541e-14 6.4536e-06]
       dfe: [12 5]
         p: [3.2541e-14 6.4536e-06]
```

We get the same correlations coefficient as in the theoretical course. The coefficients of the variables are not the same because MATLAB don't give back center reduced coefficients. Otherwise for the statistics (Wilks, etc.) we did not study them in the theoretical course.

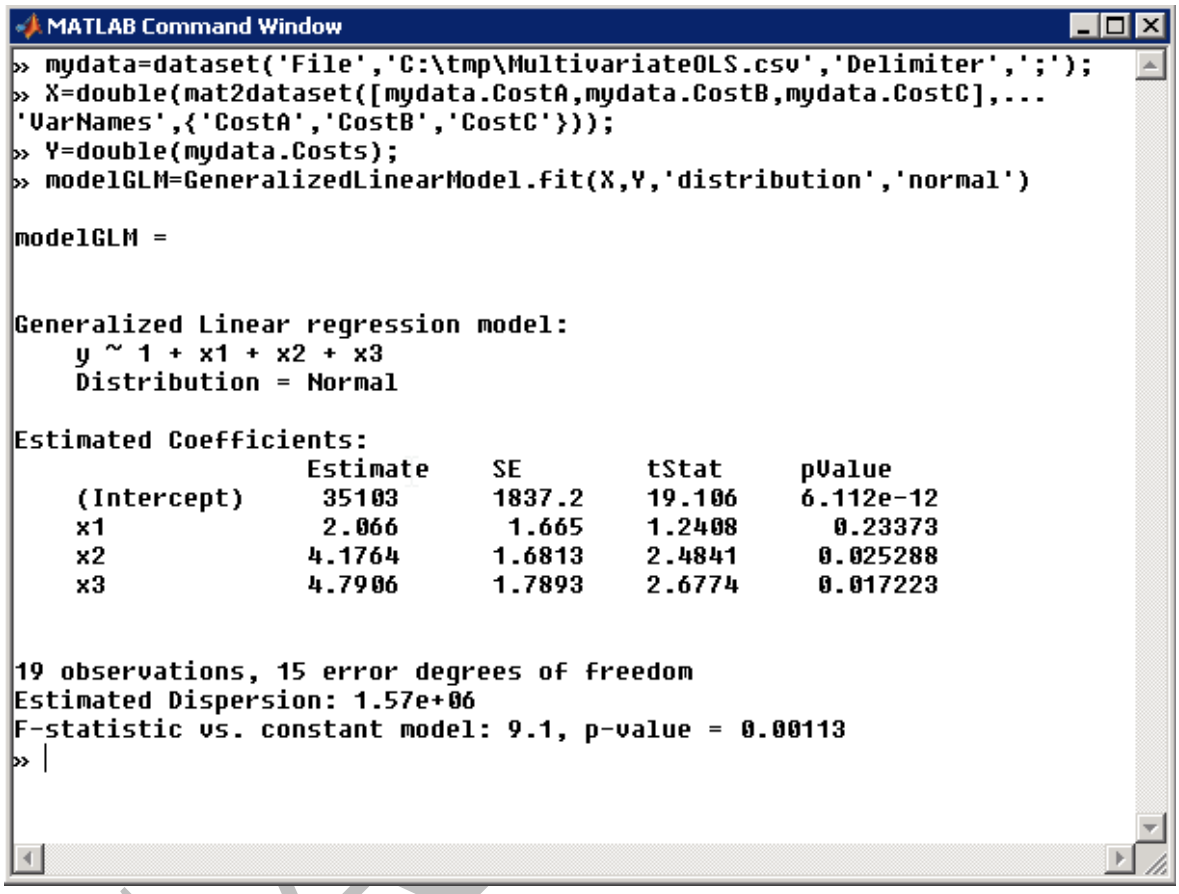

MATLAB Command Window
» plot(U(:,1),U(:,1),'.')
»
Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2

SA

Akaike Information Criterion (AIC) and Bayesian Information Criterion

The purpose here is to see if we obtain the same results as in the theoretical course and also to compare the results with R and Minitab.

First we do a "simple" classic Normal GLM of our know dataset:



At this point we have exactly the same values as Minitab and R. Now we will ask for the AIC and BIC value of this model:



As we can see... the value is quite different from R and Minitab (almost 20% of difference). We can't compare with calculations made by hand because in the theoretical course during the study of entropy we saw that the foundations of the AIC are very empirical.

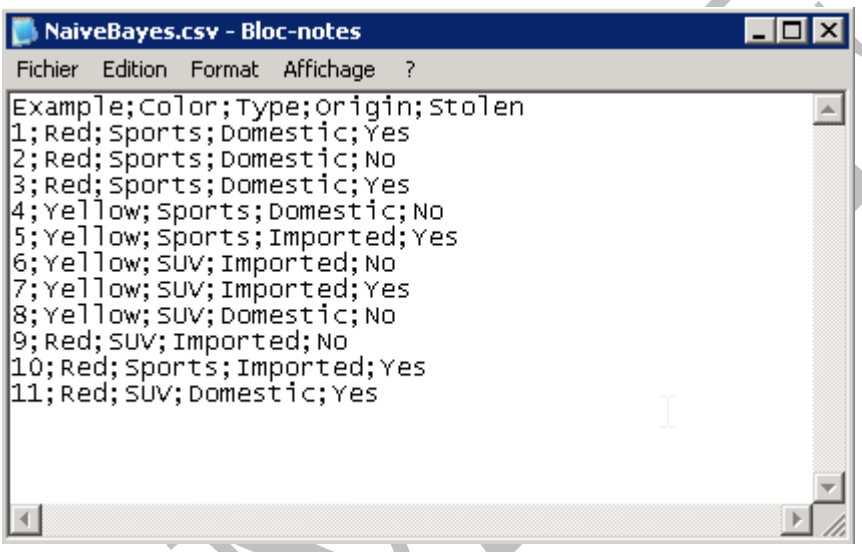
SAMPLE

Data Mining/Machine Learning

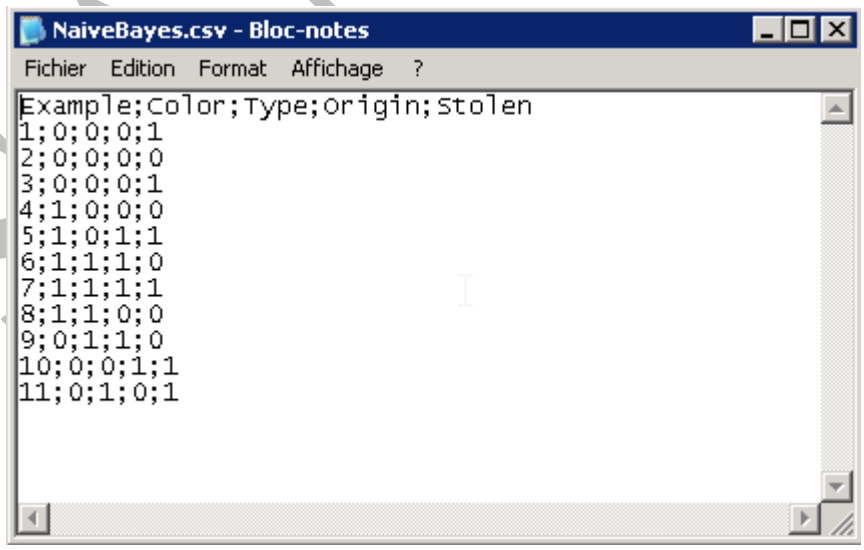
As we saw in the Numerical Methods theoretical course Data Mining/Machine learning is a very big subject. Even if my favorite software for this purpose remains at this day Tanagra first and R in second and RapidMiner in the third position let us have a look to the possibility of some MATLAB core functions in this area.

Naive Bayes

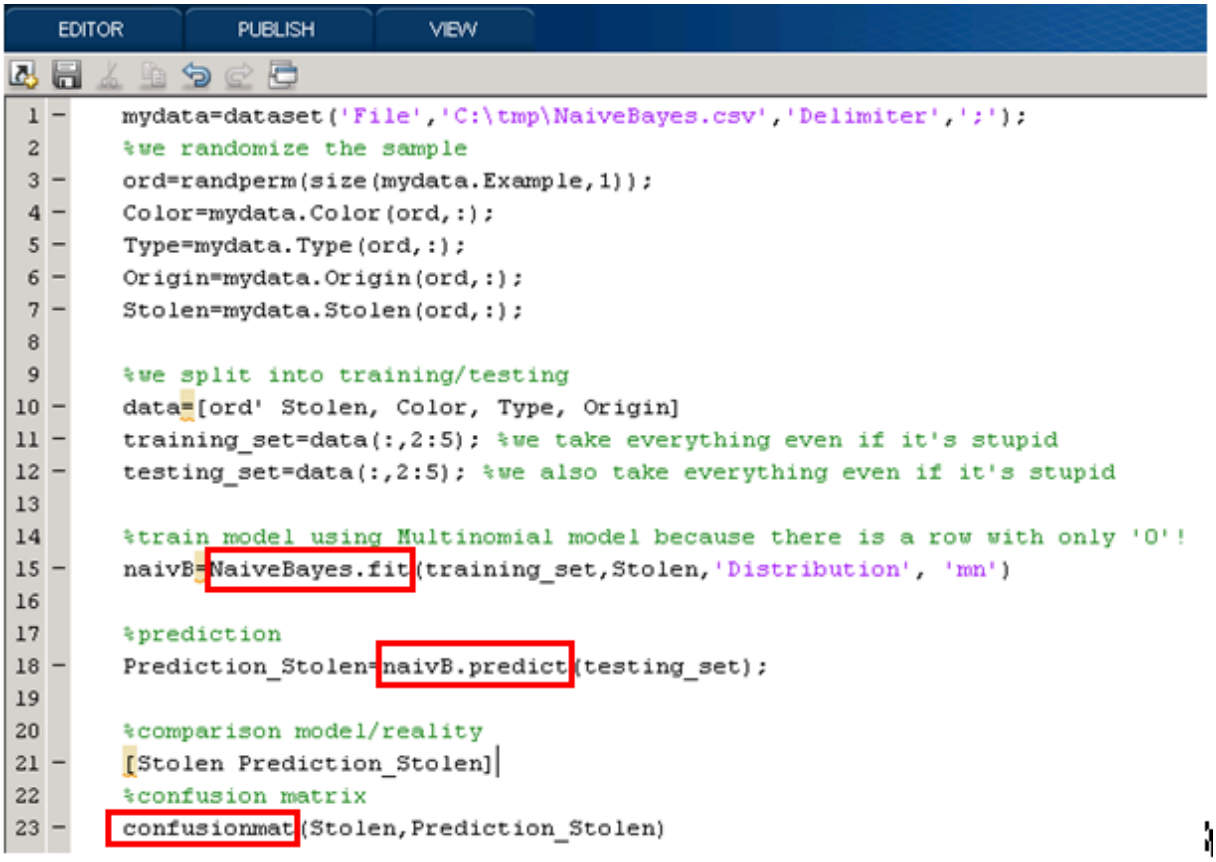
Now have a look to the naive Bayes machine learning algorithm to check also if we get the same results as in the theoretical course. For this we will use the same data:



We must convert into numeric because MATLAB doesn't handle (actually...) strings:



Now we use the **NaiveBayes.fit( )** and **NaiveBayes.predict( )** function after importing, randomizing and sampling the original data:



We use after the property **predict** and the function **confusionmat( )** to see how performs the algorithm:

SAMPLE

```
Command Window

data =

     6     0     1     1     1
     2     0     0     0     0
    10     1     0     0     1
     3     1     0     0     0
     9     0     0     1     1
    11     1     0     1     0
     4     0     1     0     0
     8     0     1     1     0
     1     1     0     0     0
     7     1     1     1     1
     5     1     1     0     1


naivB =

Naive Bayes classifier with 2 classes for 4 dimensions.
Feature Distribution(s):mn
Classes:0, 1
```

```
ans =

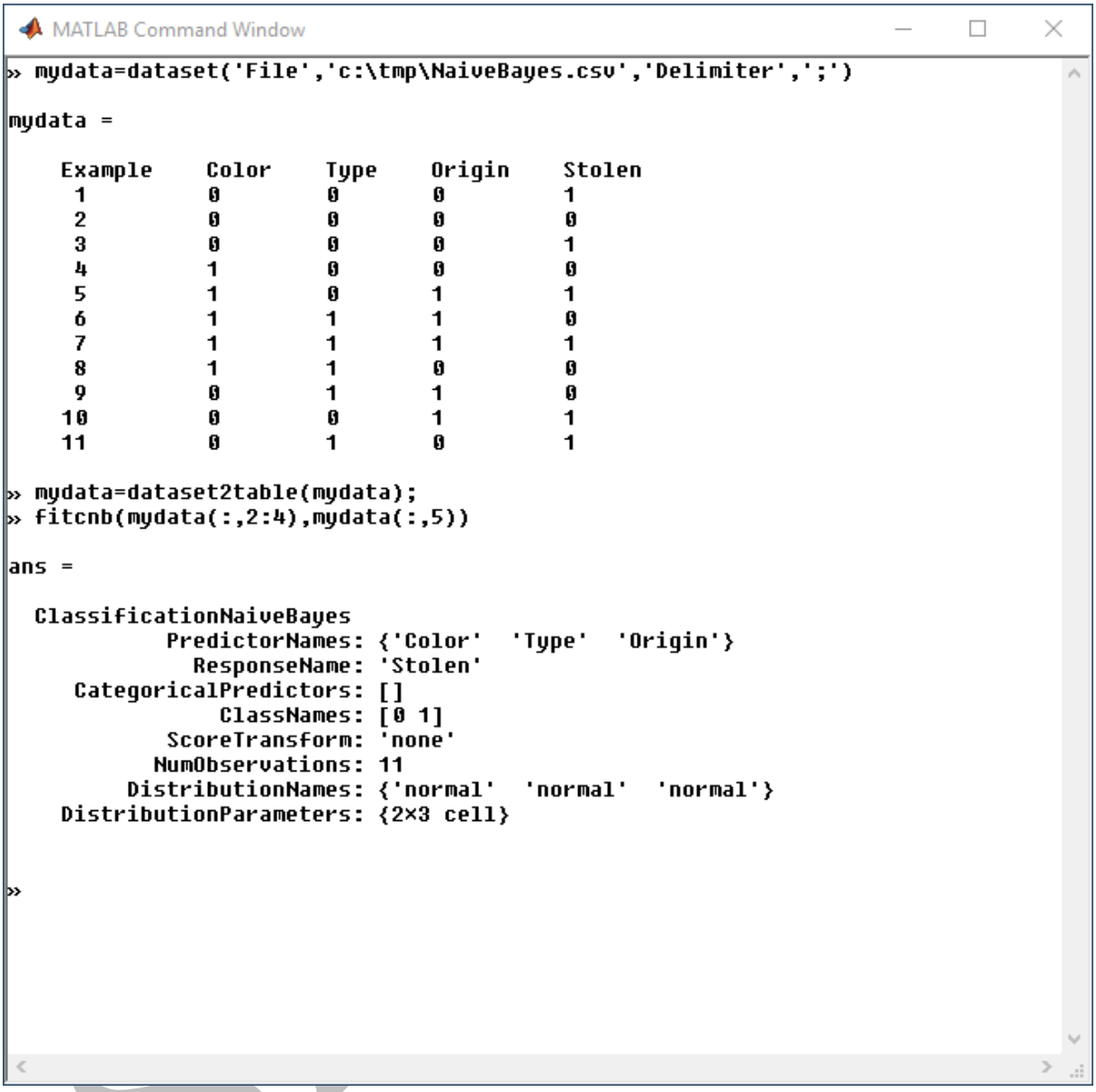
     0     0
     0     1
     1     1
     1     1
     0     0
     1     1
     0     0
     0     0
     1     1
     1     1
     1     1


ans =

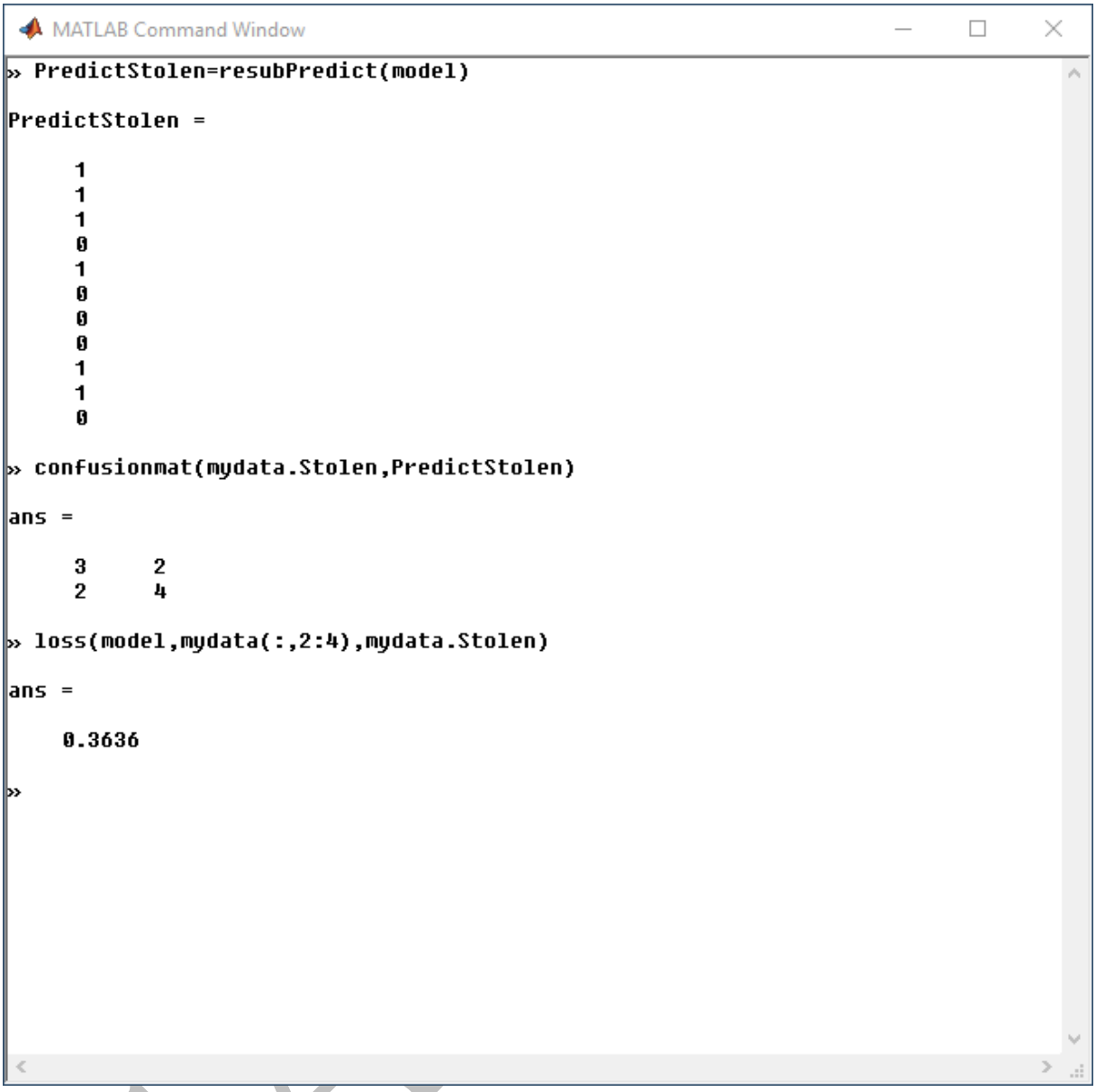
     4     1
     0     6
```

We can't compare with the theoretical course because of the sampling but it is difficult to imagine that anything can be false...

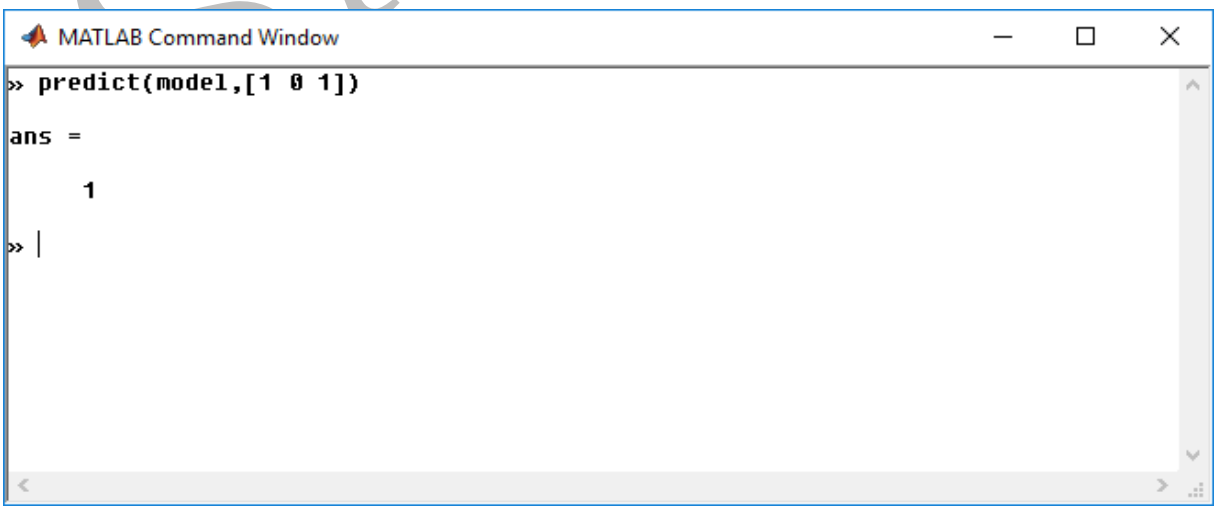
The syntax has change quite a bit since MATLAB 2017a, we have to use the function **fitcnb( )**:



and to see how the model is accurate relatively t our training set we will use now the function resubPredict( ), and the function **loss( )** if we want to get the percentage of bad accuracy of the model:

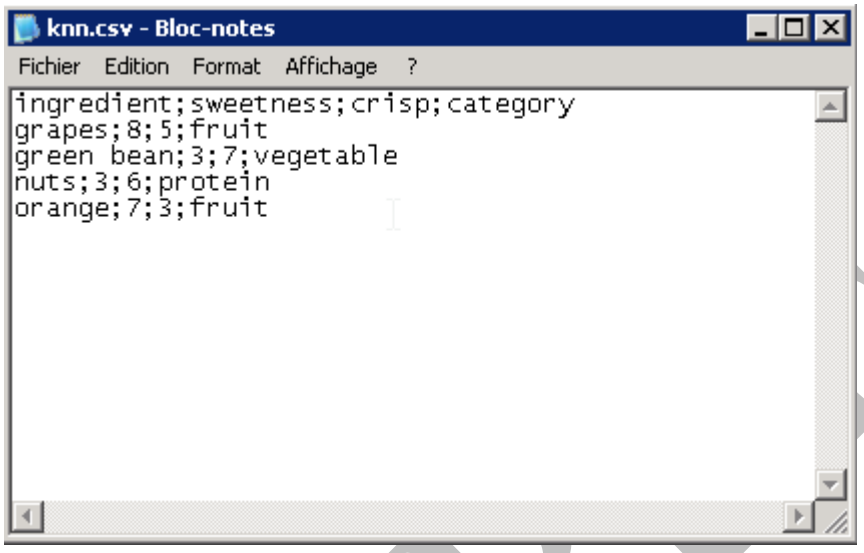


and to predict for a new value we will use the function **predict( )**:

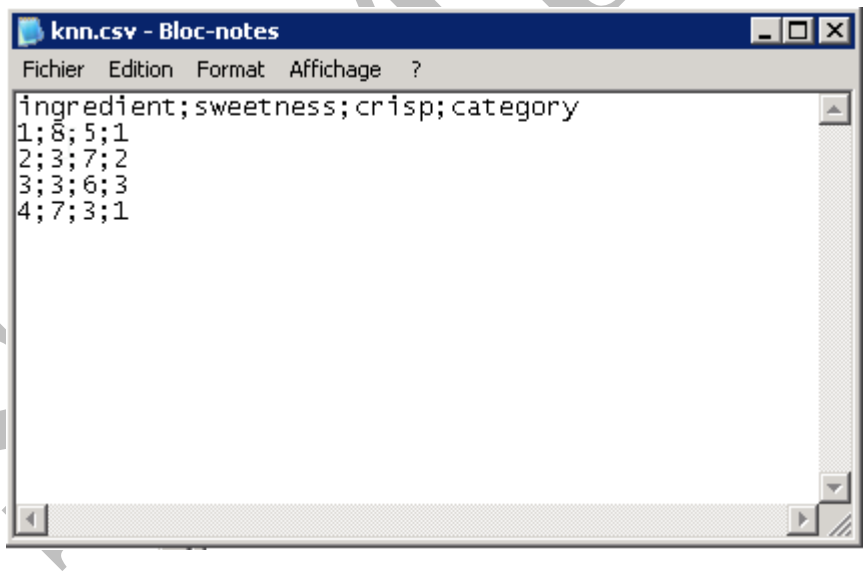


k-NN (k-Nearest Neighbors)

K Nearest Neighbors is another Machine Learning technic we have study in the theoretical course. We will also use the same small dataset to control if we get the same results as the one calculated by hand:



We must first convert all non-numeric values into numeric because MATLAB doesn't handle (actually...) string:

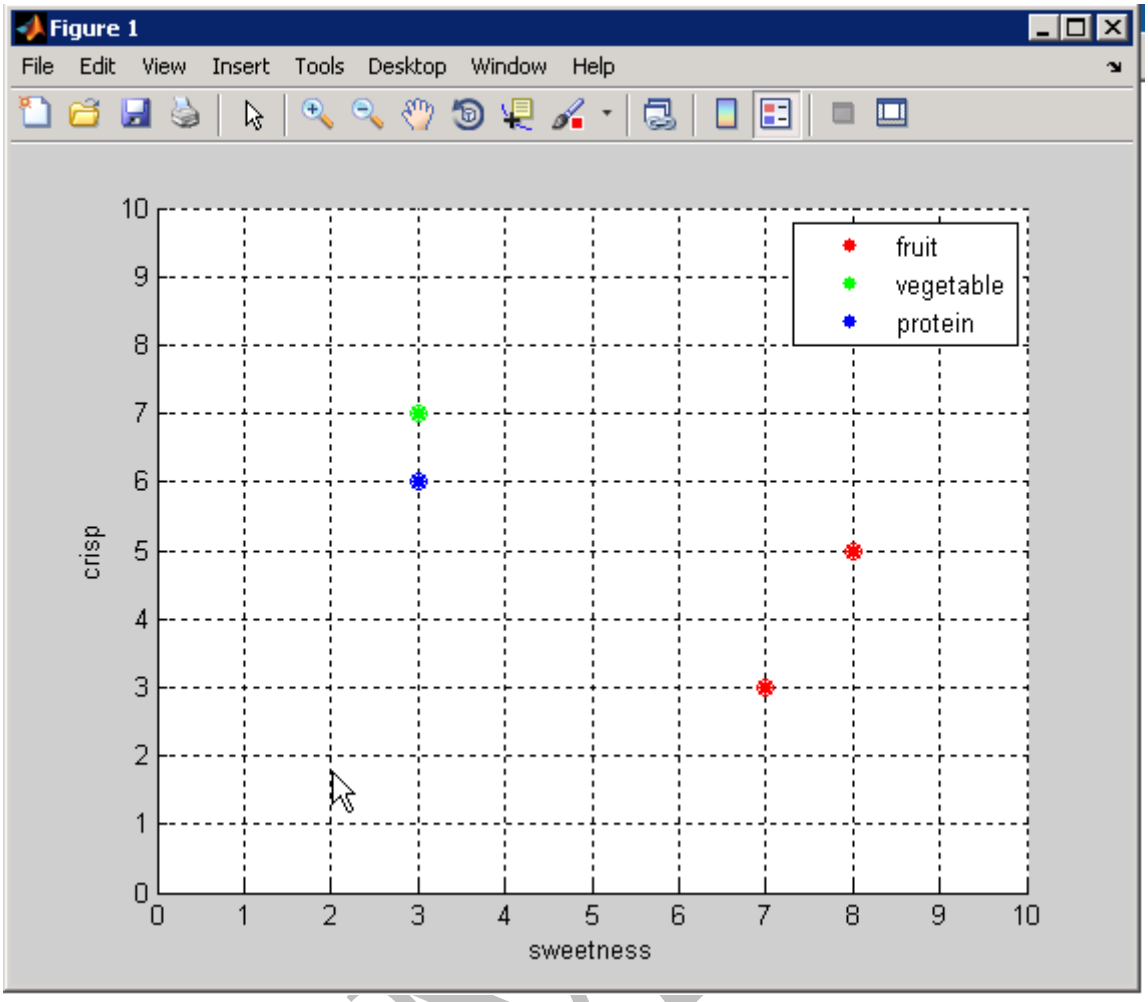


After we just need to call the function **knnclassify( )** and we plot on the same chart the model vs the original (in circles):

EDITOR PUBLISH VIEW

```
1     %we will supposed that the content is alread randomized
2 -   mydata=dataset('File','C:\tmp\knn.csv','Delimiter',';');
3     %we won't do a training and test set (see Naive Bayes example for this)
4 -   training_data=[mydata.sweetness mydata.crisp];
5
6     %with the sweetness 6 and crisp 0 classification?
7 -   number_nn=1;
8 -   knnclassify([6 0],training_data,mydata.category,number_nn)
9
10    %and with all data themself?
11 -  knn=knnclassify(training_data,training_data,mydata.category,number_nn);
12 -  comparison=[mydata.category knn]
13
14    %and a plot
15 -  gscatter(training_data(:,1),training_data(:,2),mydata.category);
16 -  xlim([0 10]);ylim([0 10]);
17
18 -  hold on;grid on;
19 -  gscatter(training_data(:,1),training_data(:,2),knn,'rgb','o');
20 -  legend('fruit','vegetable','protein');
21 -  xlabel('sweetness');ylabel('crisp');
```

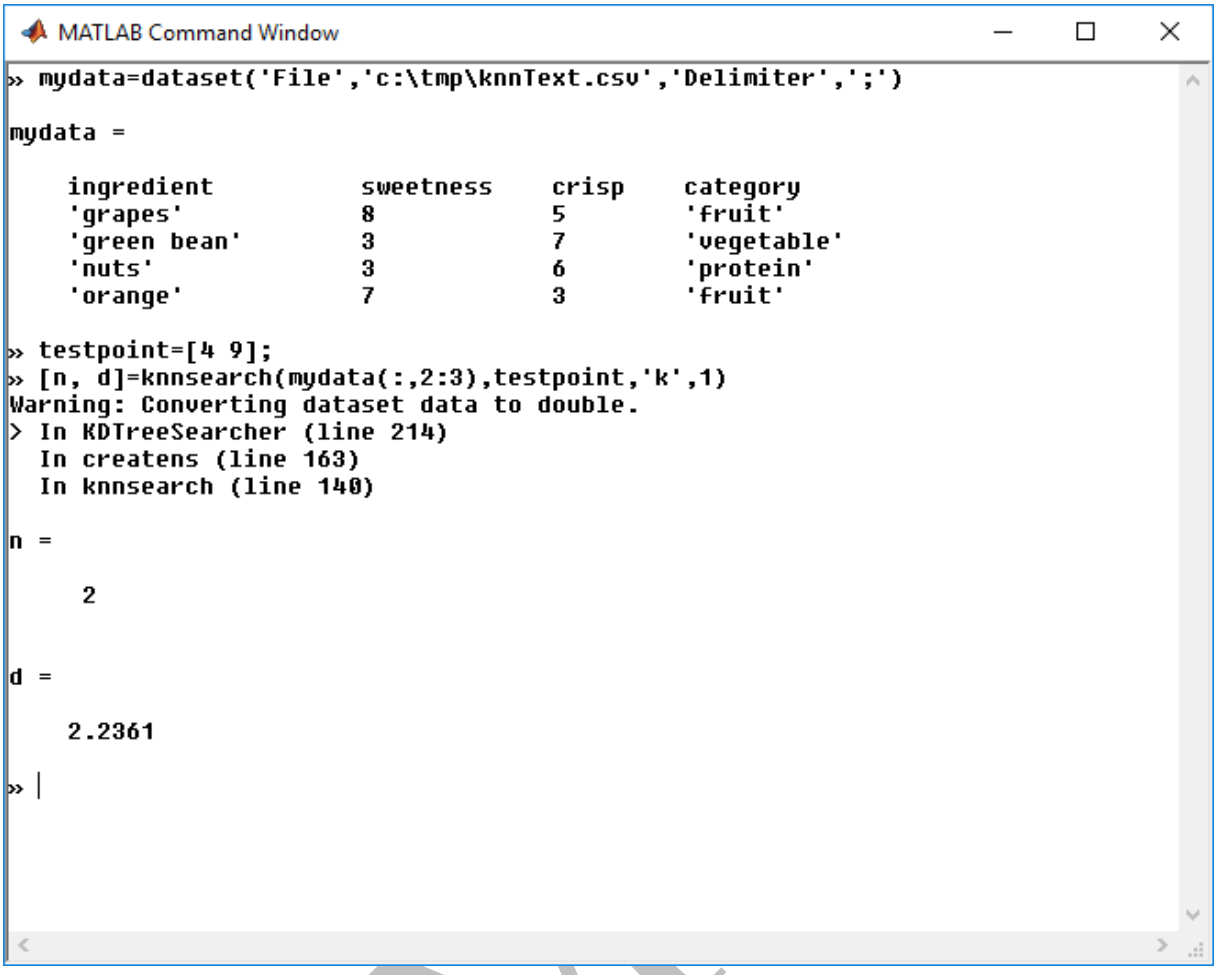
SAMPLE



As we can see the prediction model is perfect in this case.

Since MATLAB 2017a the syntax has change quite a bit using the function **knnsearch( )**:

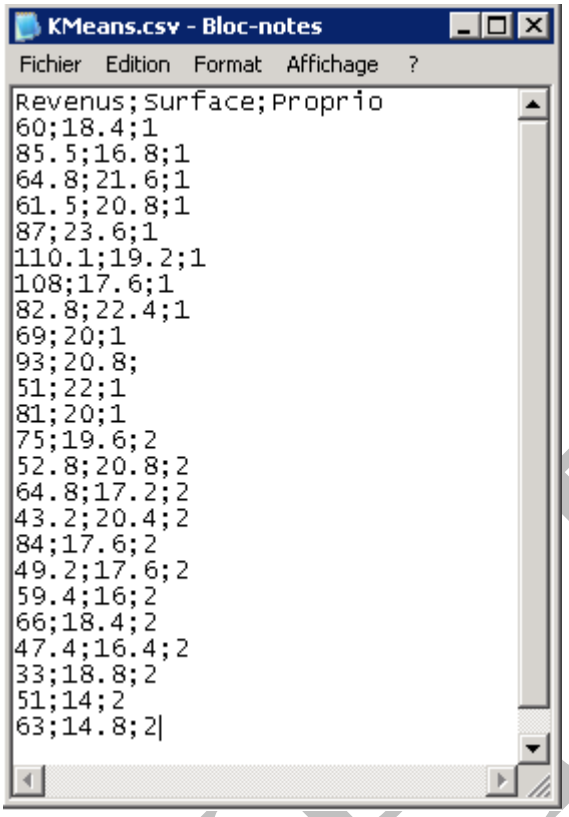
SAM



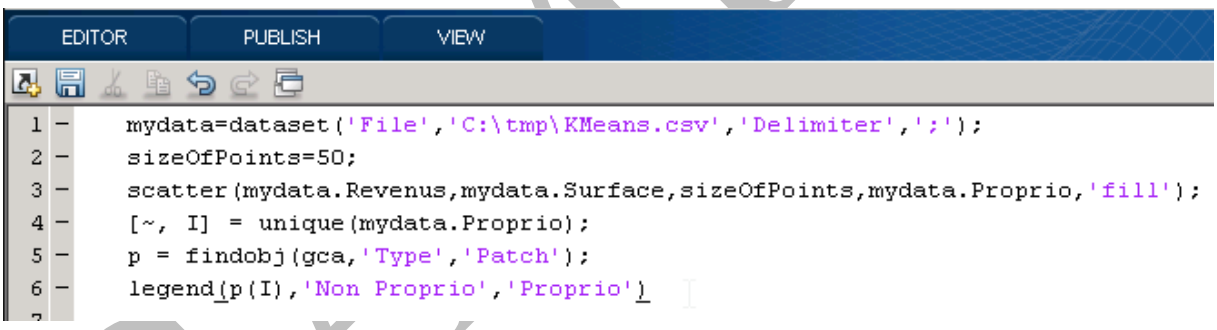
K-means (Euclidian full)

The k-means algorithm is the most easy machine learning algorithm implemented in MATLAB. To see if we found with MATLAB the same results as calculations made by hand and with Minitab, Tanagra and R we will use the same dataset:

SAMPLE

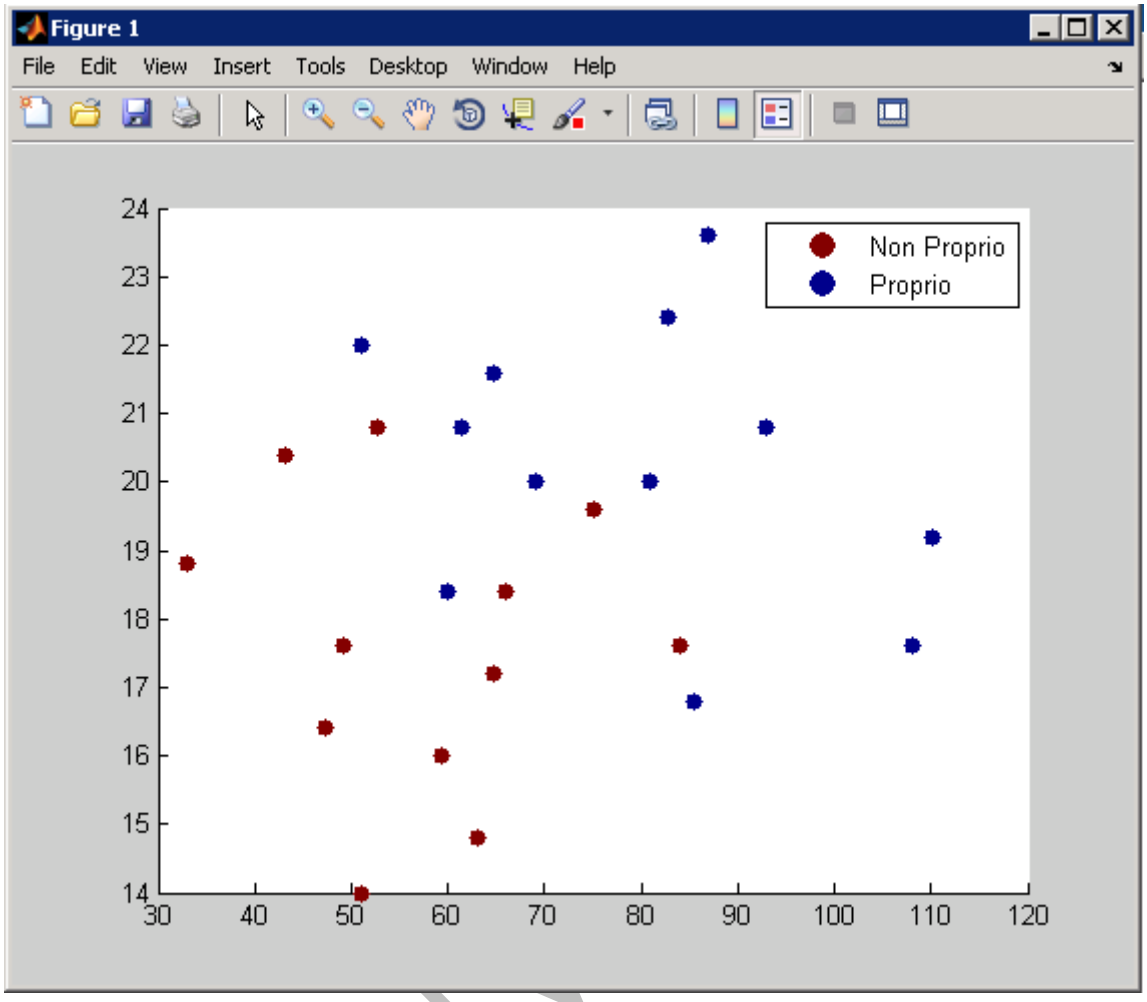


First we simply plot the actual values of the dataset:



SAMPLE

We can guess that a pattern exists:

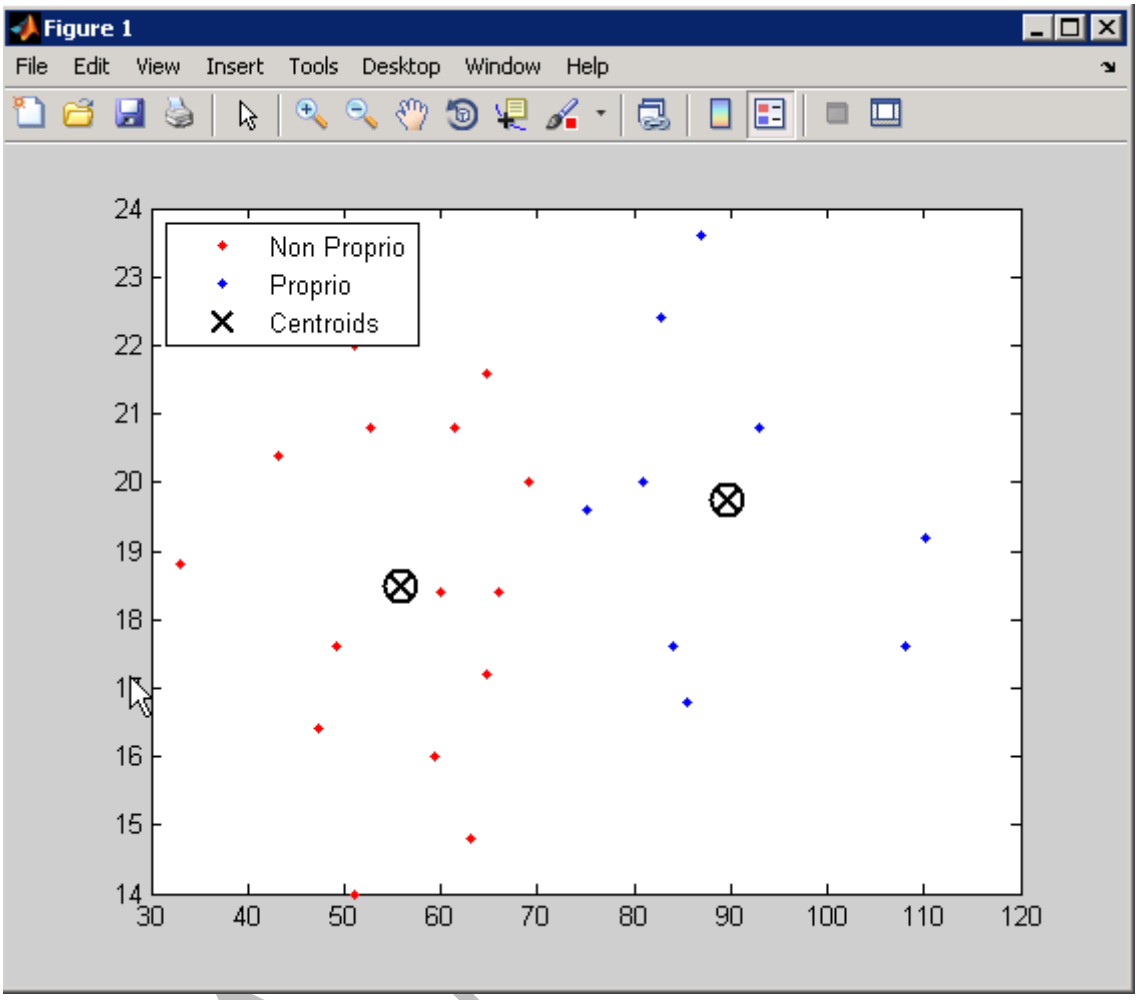


Now we use the **kmeans( )** with implicitly the Euclidian distance to found two clusters:

EDITOR PUBLISH VIEW

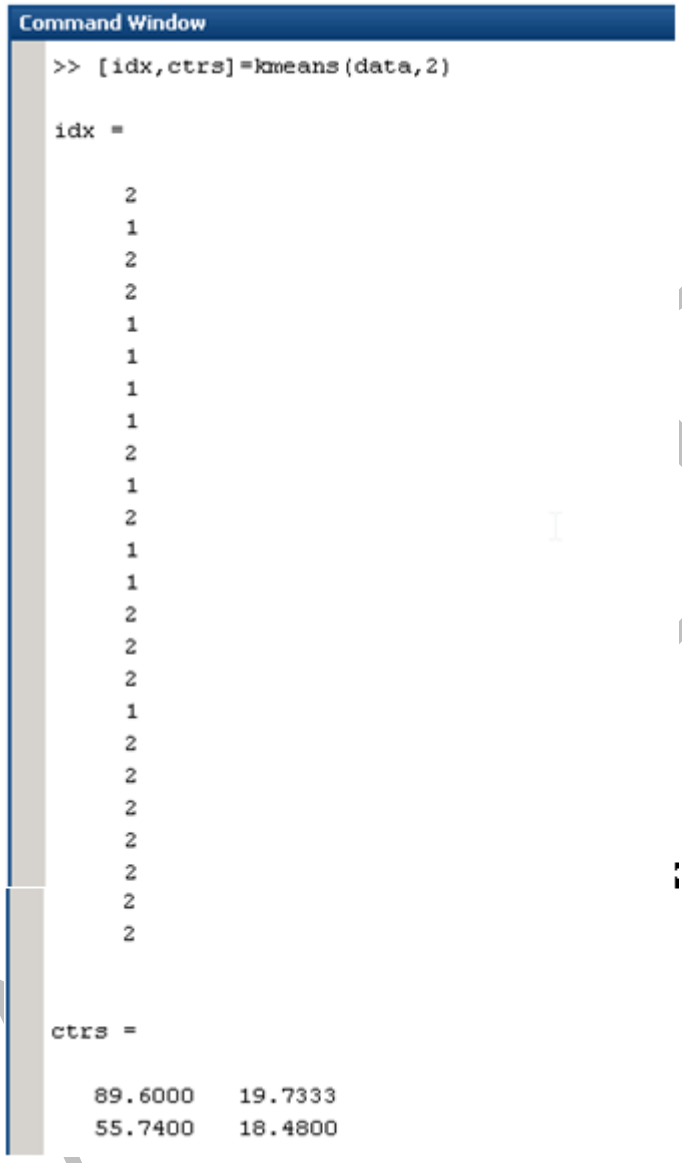
```
1  mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2  sizeOfPoints=50;
3  scatter(mydata.Revenus,mydata.Surface,sizeOfPoints,mydata.Proprio,'fill');
4  [~, I] = unique(mydata.Proprio);
5  p = findobj(gca,'Type','Patch');
6  legend(p(I),'Non Proprio','Proprio')
7
8  Clusters=2
9  data=[mydata.Revenus,mydata.Surface];
10 [idx,ctrs]=kmeans(data,Clusters)
11 plot(data(idx==1,1),data(idx==1,2),'r.','MarkerSize',12)
12 hold on
13 plot(data(idx==2,1),data(idx==2,2),'b.','MarkerSize',12)
14 plot(ctrs(:,1),ctrs(:,2),'kx','MarkerSize',12,'LineWidth',2)
15 plot(ctrs(:,1),ctrs(:,2),'ko','MarkerSize',12,'LineWidth',2)
16 legend('Non Proprio','Proprio','Centroids','Location','NW')
17 hold off
```

and we script also plot now the two centroids with what will be consider as a *Proprio* and what will not be considered as a *Proprio*:



SAM

Another thing we can do is to get directly the classification from the original dataset and the coordinates of the centroids:



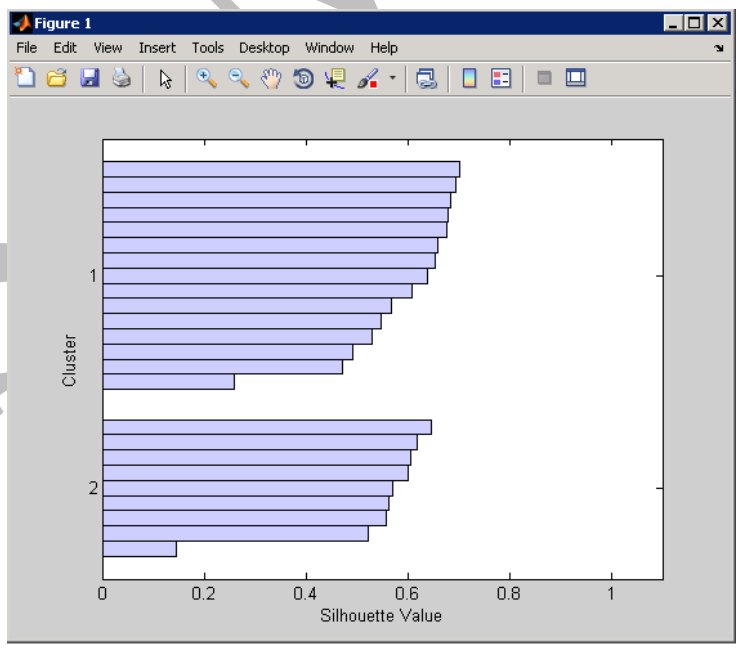
Take care!!! The order of the centroids will change randomly at each execution!

ToDo: Add here a confusion matrix using **confusionmat( )**

And what about the silhouette of the algorithm. For this purpose we use the function **silhouette( )**:

```
1  mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2  sizeOfPoints=50;
3  scatter(mydata.Revenus,mydata.Surface,sizeOfPoints,mydata.Proprio,'fill');
4  [~, I] = unique(mydata.Proprio);
5  p = findobj(gca,'Type','Patch');
6  legend(p(I),'Non Proprio','Proprio')
7
8  Clusters=2
9  data=[mydata.Revenus,mydata.Surface];
10 [idx,ctrs]=kmeans(data,Clusters)
11 plot(data(idx==1,1),data(idx==1,2),'r.','MarkerSize',12)
12 hold on
13 plot(data(idx==2,1),data(idx==2,2),'b.','MarkerSize',12)
14 plot(ctrs(:,1),ctrs(:,2),'kx','MarkerSize',12,'LineWidth',2)
15 plot(ctrs(:,1),ctrs(:,2),'ko','MarkerSize',12,'LineWidth',2)
16 legend('Non Proprio','Proprio','Centroids','Location','NW')
17 hold off
18
19 %silhouette qualitative cluster quality segmentation
20 [silh3,h] = silhouette(data,idx,'city');
21 set(get(gca,'Children'),'FaceColor',[.8 .8 1])
22 xlabel('Silhouette Value')
23 ylabel('Cluster')
```

And we get:



Each bar represents the affectation of a point to the centroid and the height of the bar represents its normalized distance.

We can check that the system use the Euclidian distance by calculating yourself the affectation of each point at each centroid:

```
25      %---------------------------------
26      %prediction with all data (take care again! the cluster center number can
27      %also change at each run!) what is the nearest cluster id!
28 -    [idx,ctrs]=kmeans(data,Clusters);
29 -    test_point=[90 20];
30 -    for i = 1:size(ctrs,2)
31 -        d(:,i) = sum((data-ctrs(repmat(i,size(data,1),1),:)).^2,2);
32 -    end
33 -    [distances,predicted] = min(d,[],2);
34      ctrs
35 -    predicted
36
```

```
Command Window
ctrs =

   89.6000   19.7333
   55.7400   18.4800

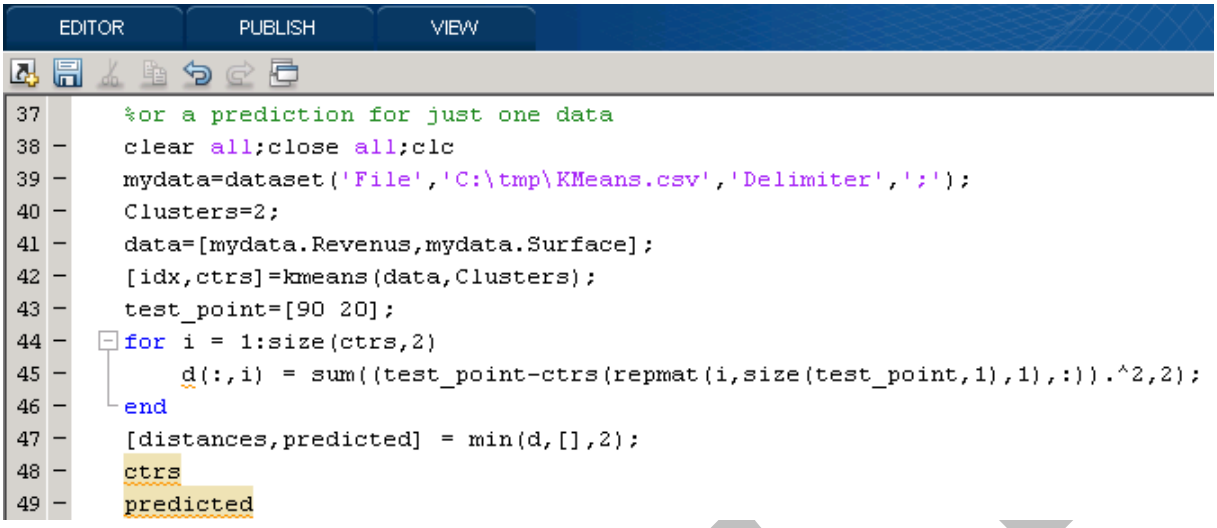

predicted =

     2
     1
     2
     2
     1
     1
     1
     1
     2
     1
     2
     1
     1
```

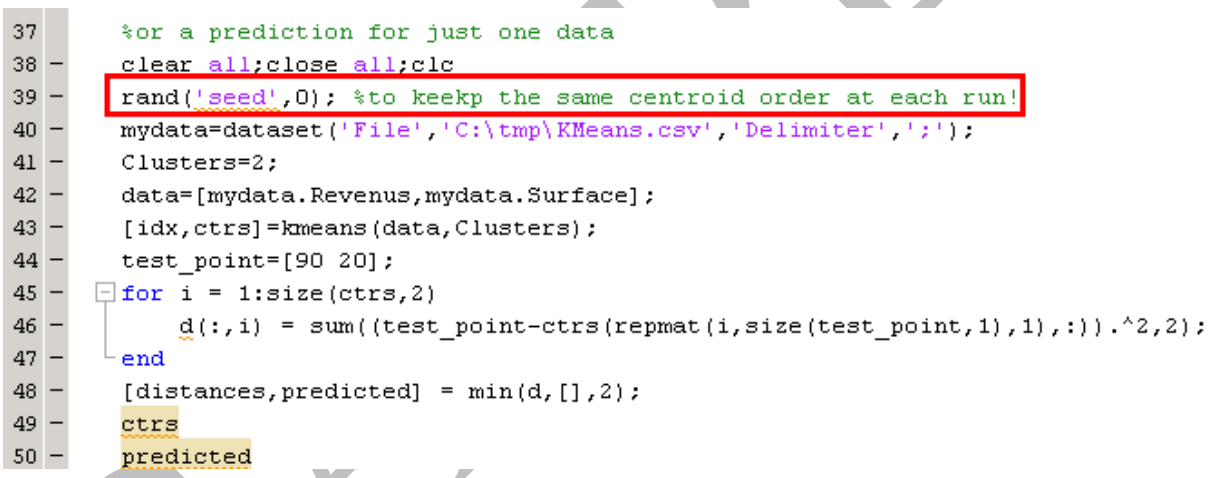
Or much more interesting: We do the something with a new point!:

```
36
37      %or for just one data
38 -    [idx,ctrs]=kmeans(data,Clusters);
39 -    test_point=[90 20];
40 -    for i = 1:size(ctrs,2)
41 -        d(:,i) = sum((test_point-ctrs(repmat(i,size(test_point,1),1),:)).^2,2);
42 -    end
43 -    [distances,predicted] = min(d,[],2);
44 -    ctrs
45 -    predicted
```

Here is the whole part script for classification of new points for those that want a global overview of the last steps:



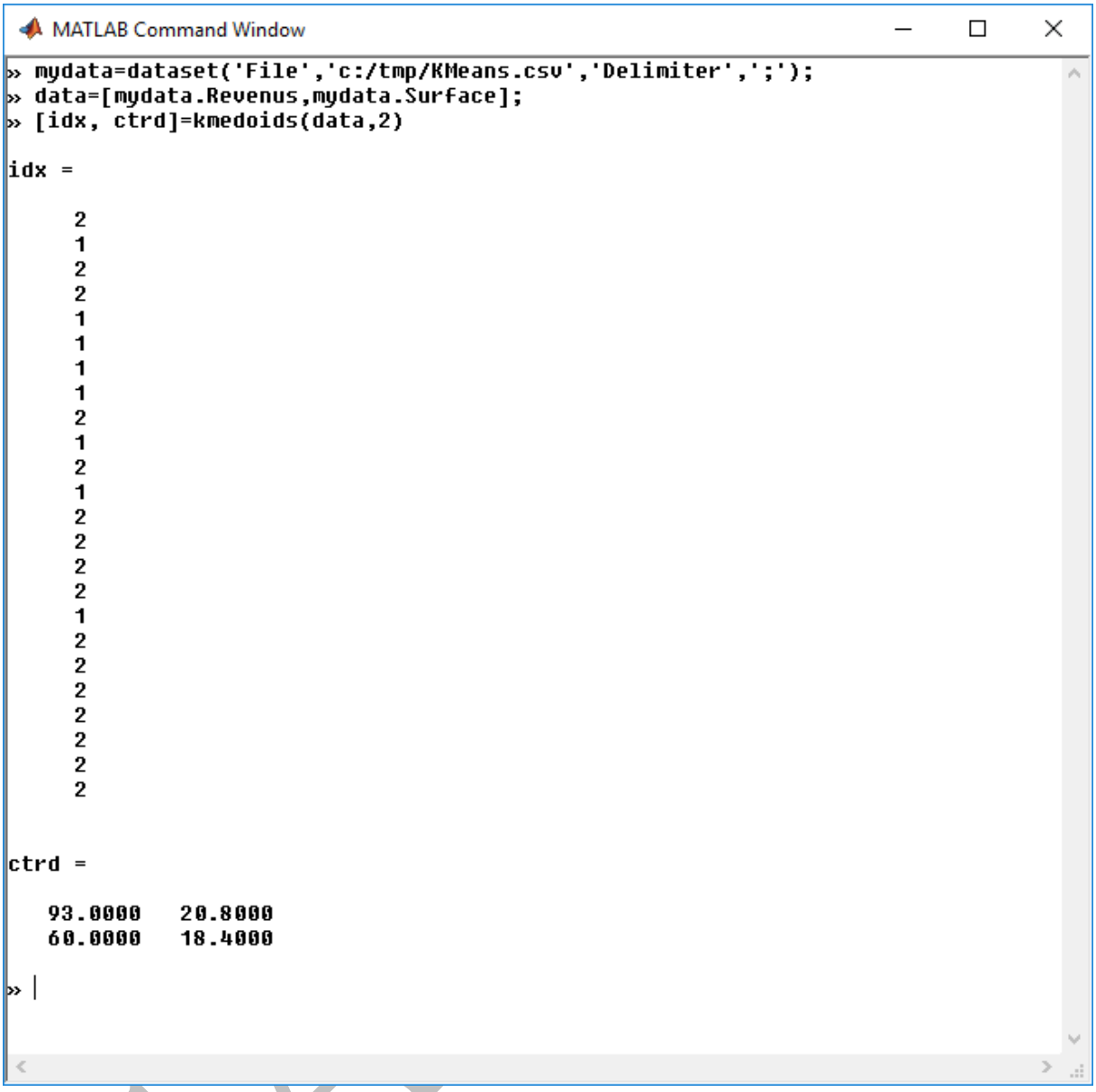
If you want to be able to reproduce the same results at each run thus seed the random generator:



K-medoids

The k-medoids is very similar to the k-means. The only difference is that the centroids are put on the nearest physical point of the initial dataset.

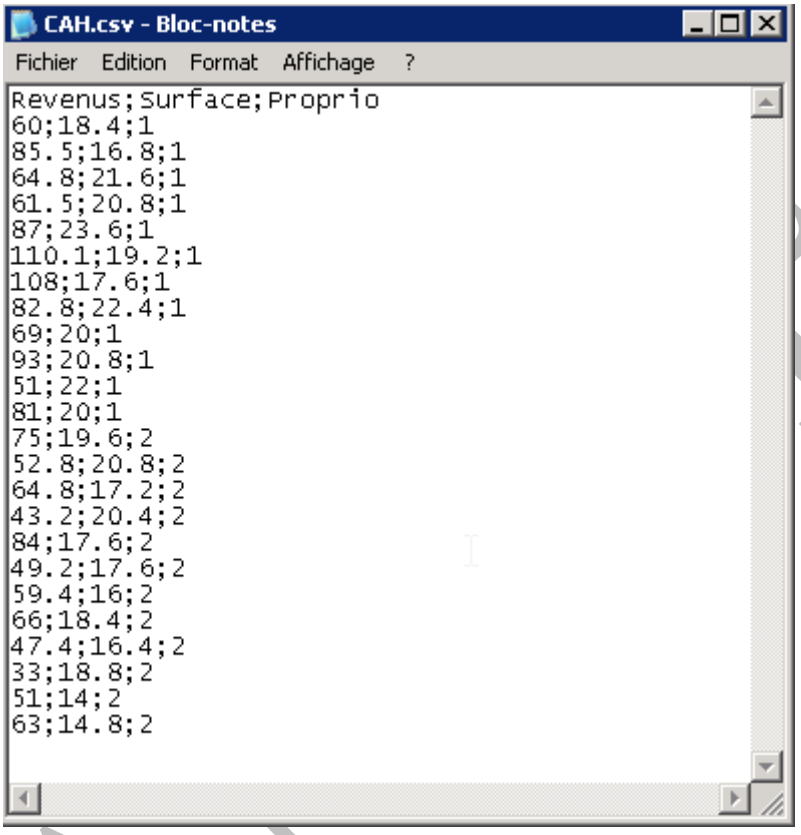
The only one thing that has to be know is the function name as everything else follows the same procedure as for the k-means:


MATLAB Command Window
» mydata=dataset('File','c:/tmp/KMeans.csv','Delimiter',';');
» data=[mydata.Revenus,mydata.Surface];
» [idx, ctrd]=kmedoids(data,2)
idx =
2
1
2
2
1
1
1
1
2
1
2
1
2
2
2
2
1
2
2
2
2
2
2
2
ctrd =
93.0000 20.8000
60.0000 18.4000
»

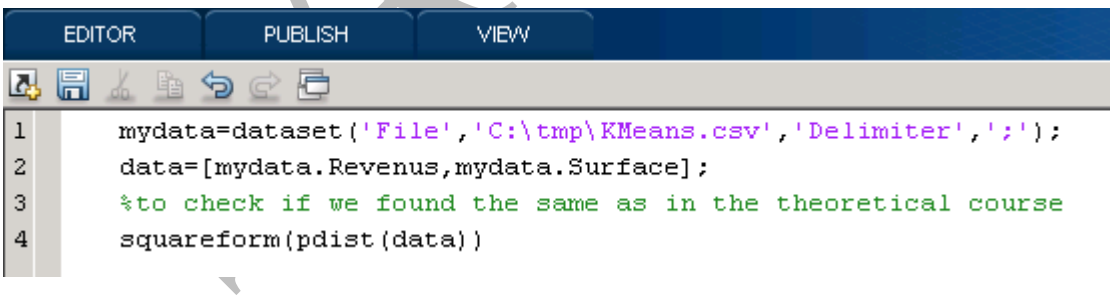
Hierarchical Ascendant Classification (HAC) dendrogram

CAH Dendrogram was an easy part of the Data Mining course: easy to define, boring to calculate but without using complex mathematics...

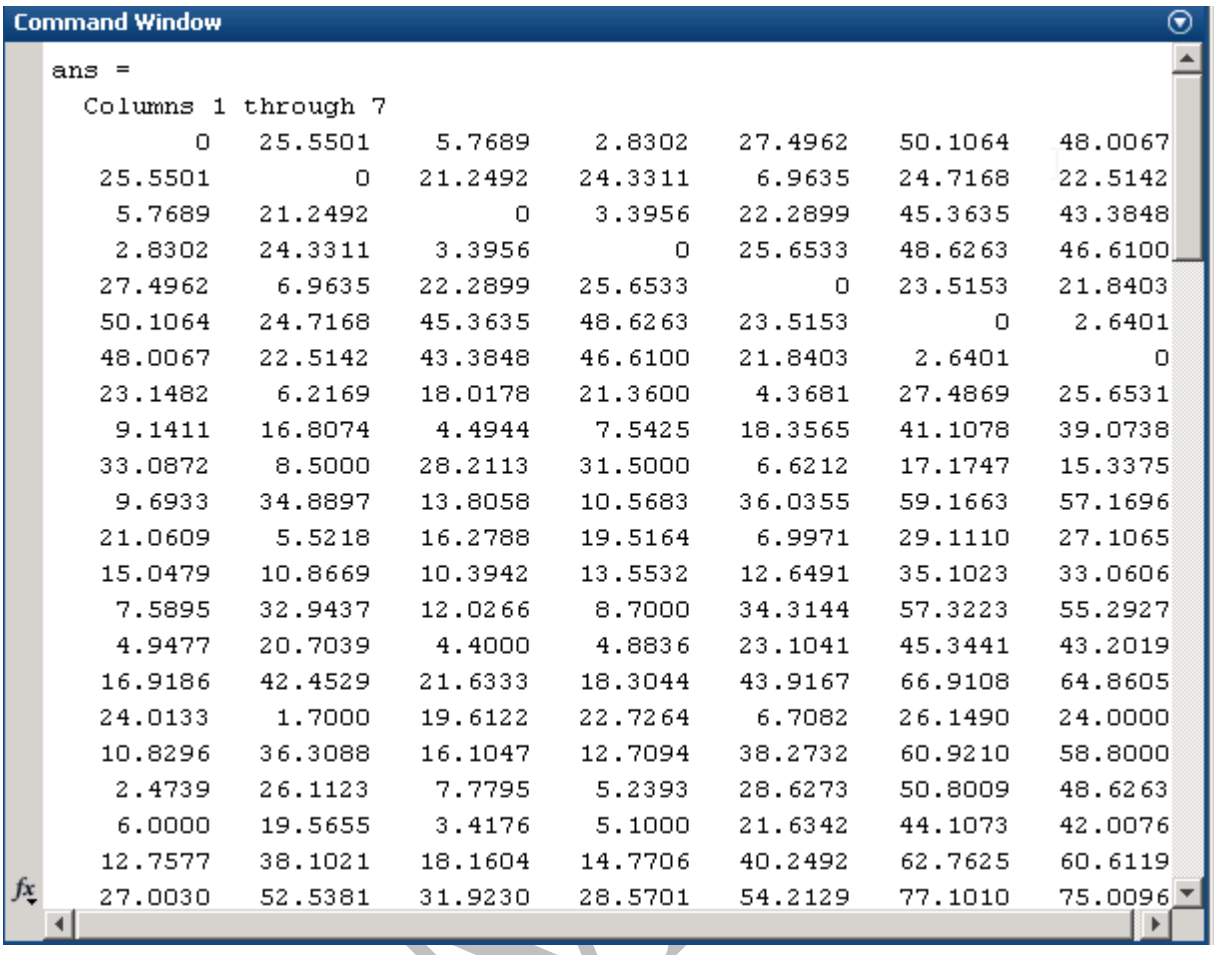
We will also use the same data as in the theoretical course



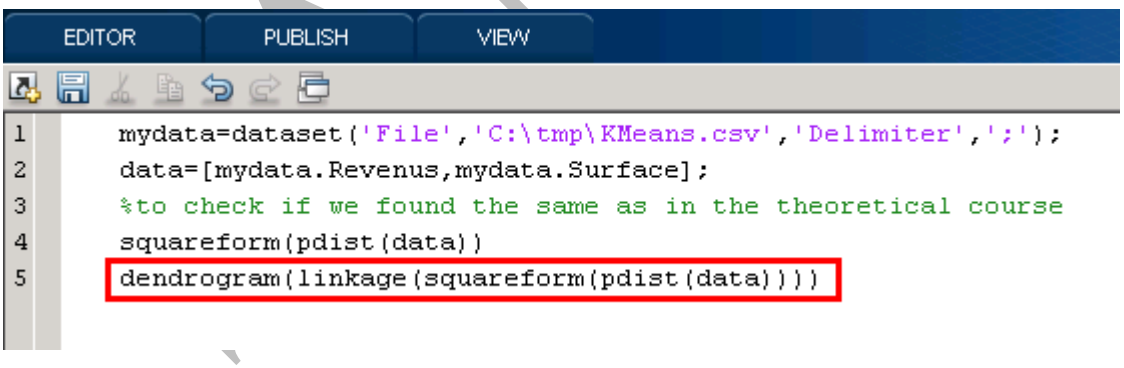
First we see that in MATLAB using the functions **pdist( )** and **squareform( )** together:



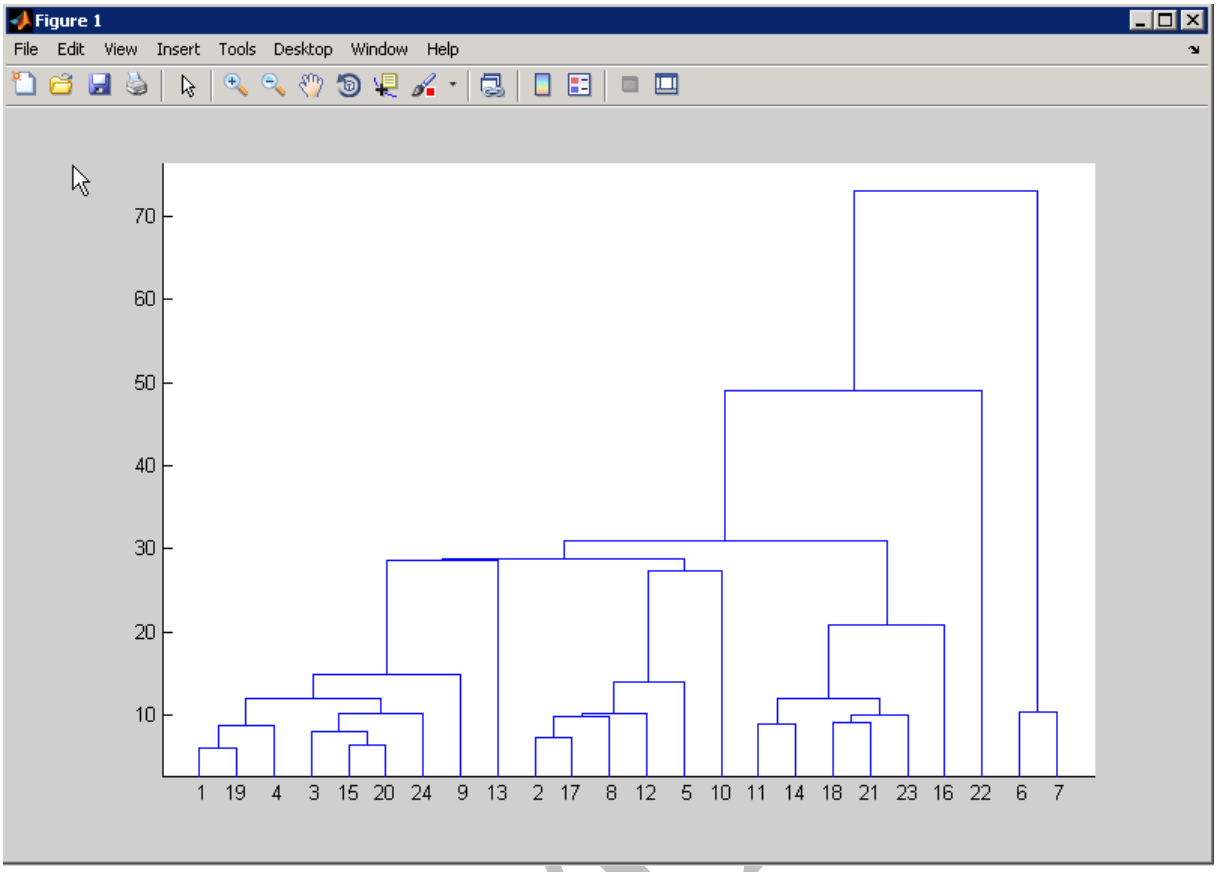
So we can see if get the same matrix as the one calculated by hand (yes this was boring...):



Now if we use the **dendrogram( )** function:



We get:



So we get the same thing as in the theoretical course! Nice!

SAMP

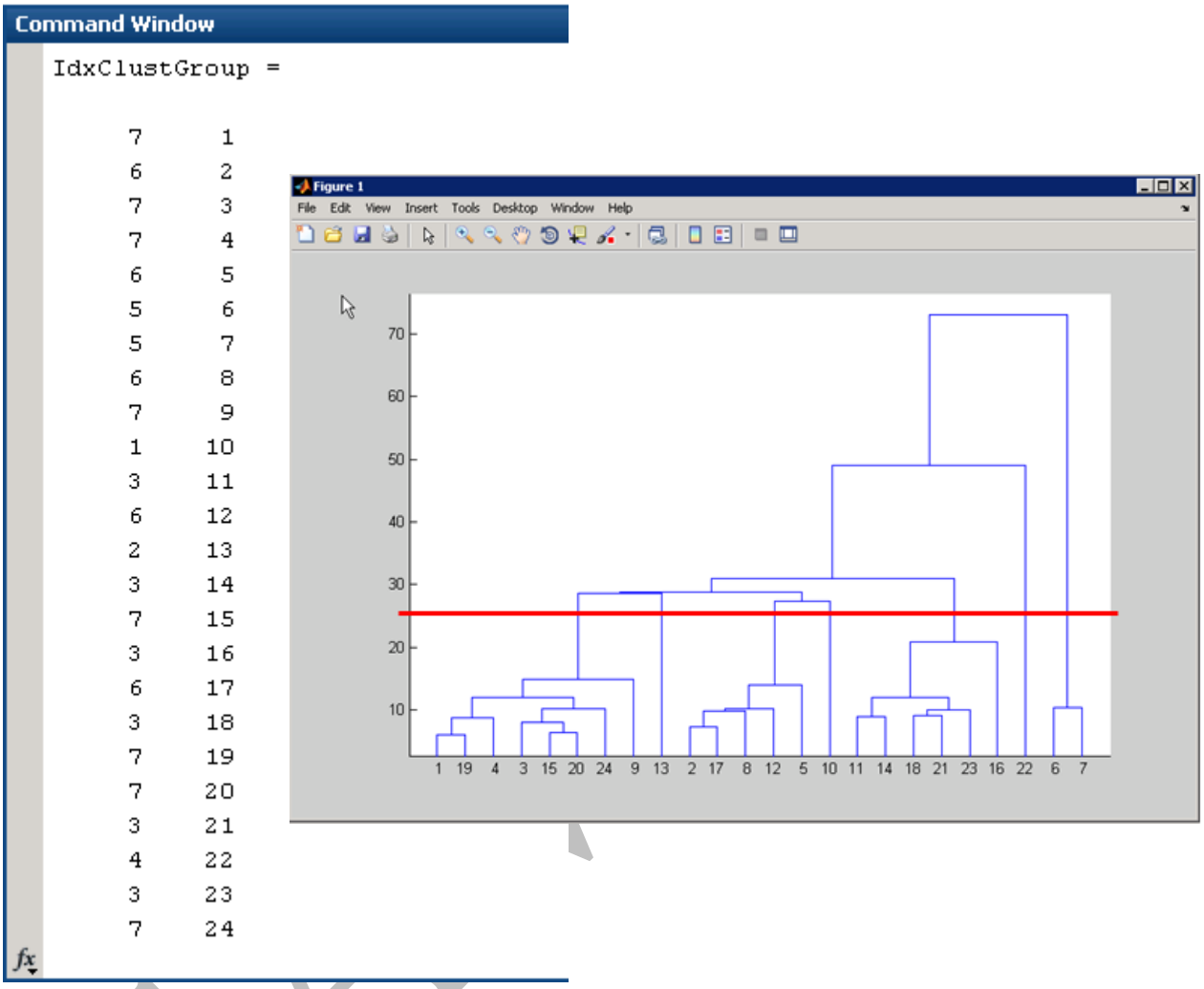
Now have a look to what we call the linkages. Each row represents a couple at the bottom of the tree. But the distance shown are a little bit strange. They do not correspond with what we get in the theoretical course (that were perfectly corresponding with the distance given by Minitab):

```
Command Window
>> linkage(squareform(pdist(data)))
ans =
    1.0000   19.0000    6.0094
   15.0000   20.0000    6.3535
    2.0000   17.0000    7.3220
    3.0000   26.0000    7.9525
    4.0000   25.0000    8.7235
   11.0000   14.0000    8.8819
   18.0000   21.0000    9.0074
    8.0000   27.0000    9.8519
   23.0000   31.0000   10.0305
   12.0000   32.0000   10.1512
   24.0000   28.0000   10.2389
    6.0000    7.0000   10.3454
   30.0000   33.0000   11.8861
   29.0000   35.0000   11.9591
    5.0000   34.0000   13.9316
    9.0000   38.0000   14.8392
   16.0000   37.0000   20.8898
   10.0000   39.0000   27.2857
   13.0000   40.0000   28.6776
   42.0000   43.0000   28.8274
   41.0000   44.0000   30.9786
   22.0000   45.0000   49.1230
   36.0000   46.0000   73.0863
```

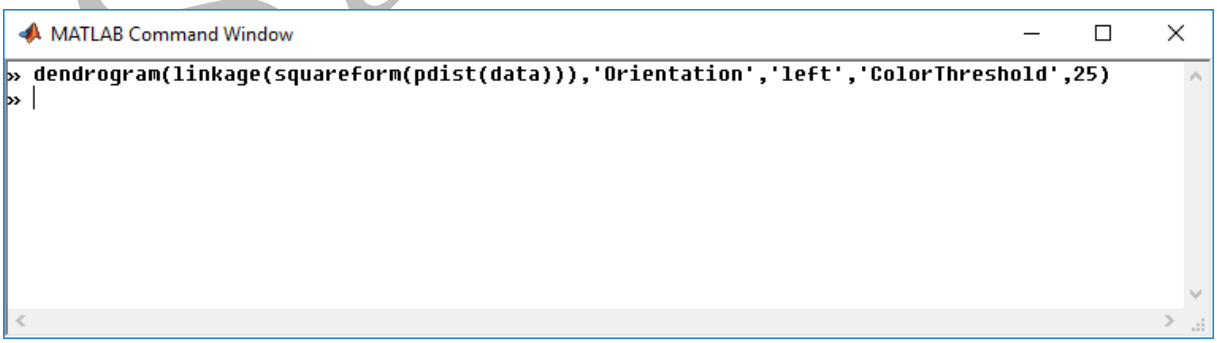
Now using the function **cluster( )** with a given **cutoff** high we can calculate row data belongs to which category:

```
EDITOR  PUBLISH  VIEW
1 -  mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';')
2 -  data=[mydata.Revenus mydata.Surface];
3 -  dendrogram(linkage(squareform(pdist(data))));
4    %cluster and cutt-off based on distance
5 -  ClustGroup=cluster(linkage(squareform(pdist(data))),'cutoff',25,...
6        'criterion','distance');
7 -  IdxClustGroup=[ClustGroup, (1:1:length(data))']
```

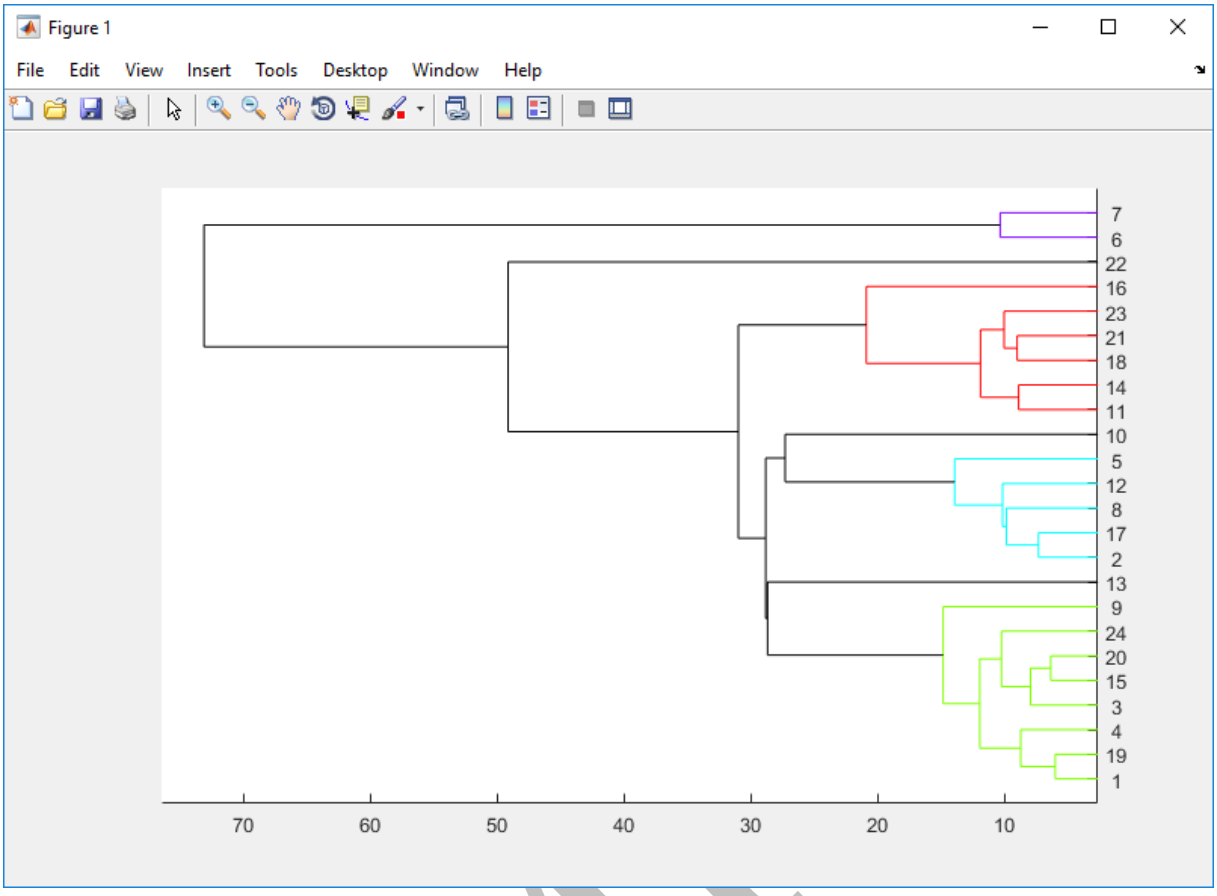
Following this cluster rule with this empirical cluster cutoff high we get (the second column contains the originals row data line number this is why there are 24 rows) the following result in the *Command Window*. Focus on the fact that at the cutoff of 25, as you can see on the chart, there are 7 clusters (seven branches) numbered for right to left:



We can also write prune visually the dendrogram to get:



To get:

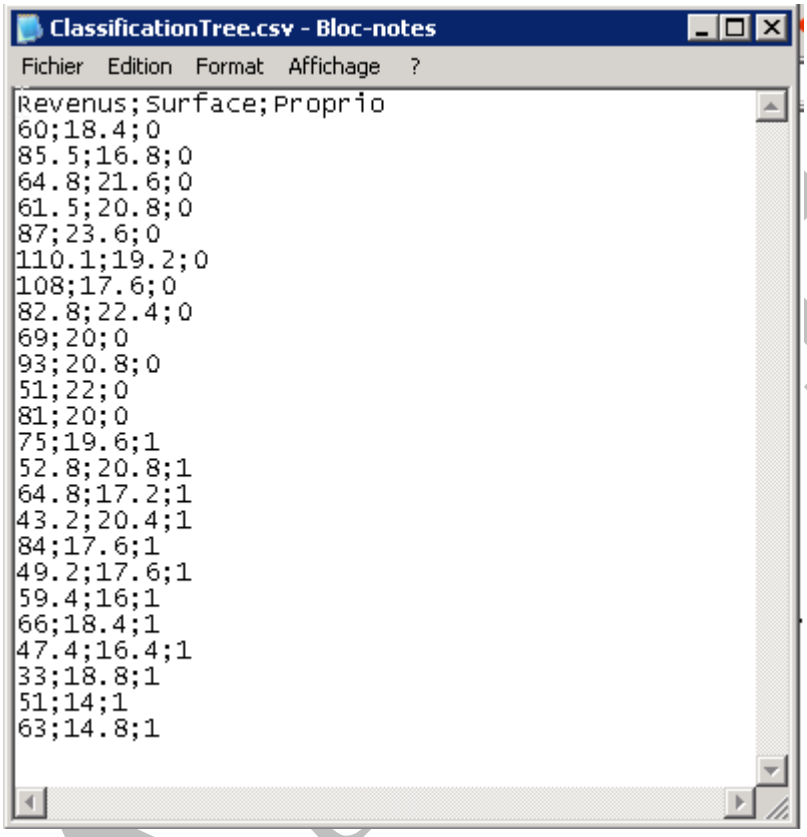

Figure 1
File Edit View Insert Tools Desktop Window Help
7
6
22
16
23
21
18
14
11
10
5
12
8
17
2
13
9
24
20
15
3
4
19
1
70
60
50
40
30
20
10

SAMP

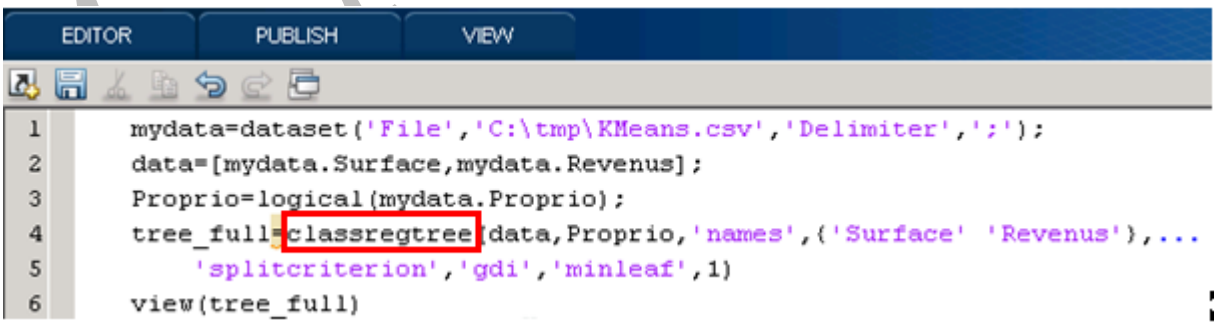
CART Classification trees (Gini index based)

In the Data Mining course we also see how to create a classification system based on a technique called "CART" (Classification And Regression Tree) using the empirical Gini index.

First we will use the same data:



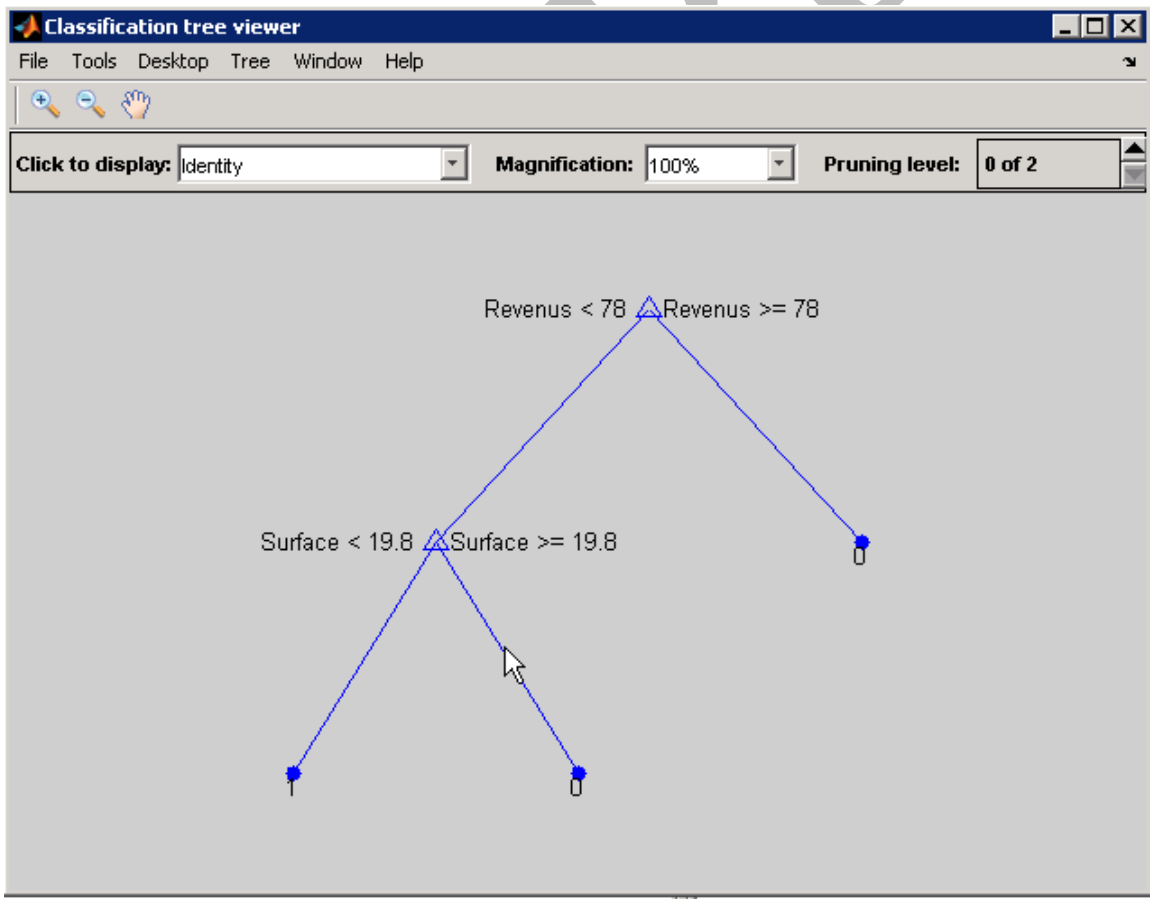
We import the data as usual in a dataset, we define the *Proprio* column as being logical (binary) with the function **logical( )** et after we run the **classregtree( )** function with the *gdi* parameter to say that we want an Gini index classification:



To see the result we use the function **view( )** on the tree object:

```
Command Window
>> mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
data=[mydata.Surface,mydata.Revenus];
Proprio=logical(mydata.Proprio);
tree_full=classregtree(data,Proprio,'names',{'Surface' 'Revenus'},...
    'splitcriterion','gdi','minleaf',1)
view(tree_full)
tree_full =
Decision tree for classification
1  if Revenus<78 then node 2 elseif Revenus>=78 then node 3 else 0
2  if Surface<19.8 then node 4 elseif Surface>=19.8 then node 5 else 1
3  class = 0
4  class = 1
5  class = 0
fx >>
```

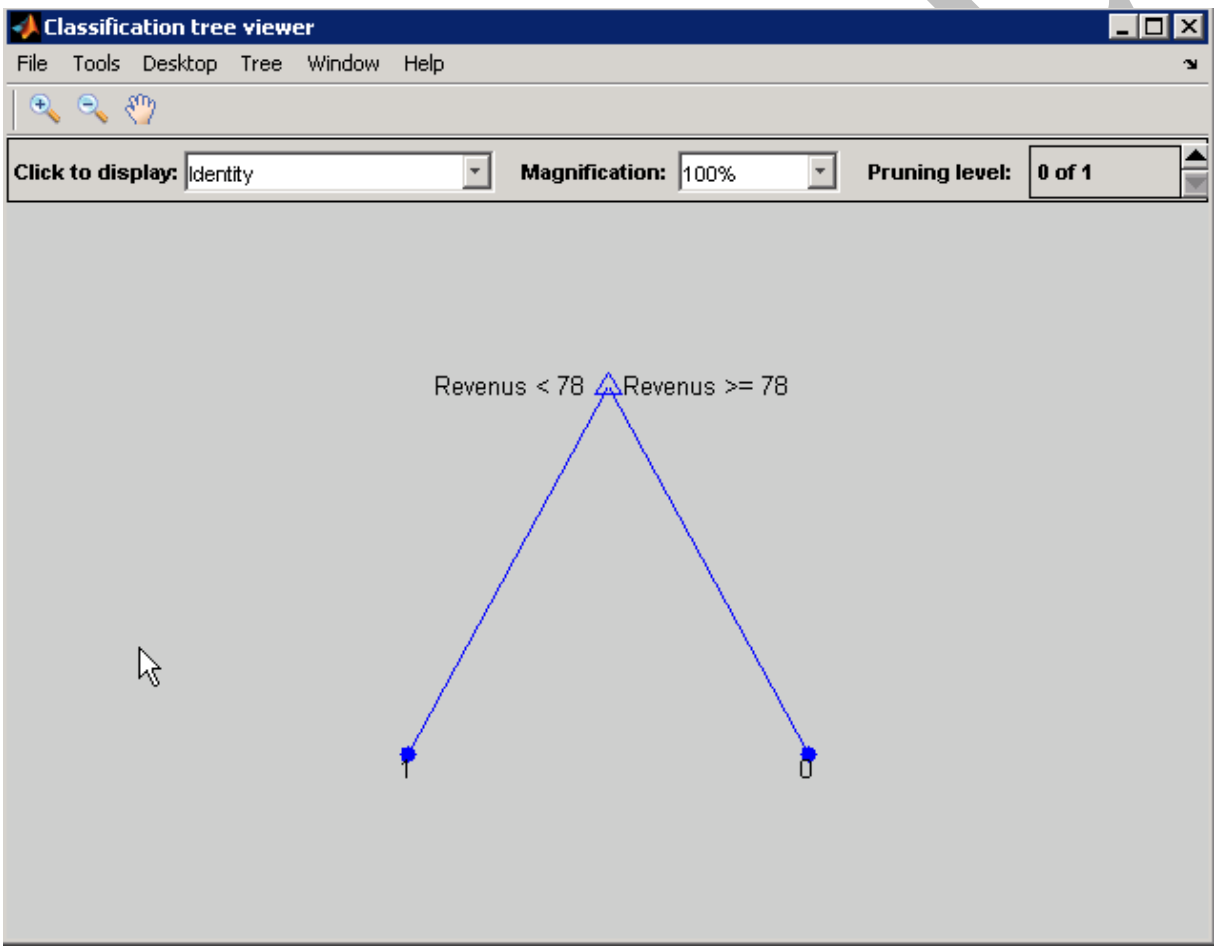
we also get the following tree:



The last value of 19.8 correspond perfectly with what we get in the theoretical course but not the second value of 78 (in the theoretical course we get 84.75)... I have to investigate where this difference comes from.

```
1   mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2   data=[mydata.Surface,mydata.Revenus];
3   Proprio=logical(mydata.Proprio);
4   tree_full=classregtree(data,Proprio,'names',{'Surface' 'Revenus'},...
5       'splitcriterion','gdi','minleaf',1)
6   view(tree_full)
7
8   %some tree prunning for fun!
9   tree_pruned = prune(tree_full,'level',1);
10  view(tree_pruned)
```

We can make tree pruning if we want:



We can also predict the classification for a new input but the given output is false in comparison to the drawed tree above and I must investigate why!:

```
Command Window
>> eval(tree_full,[75 19])

ans =

    'O'

>>
```

But for the whole dataset we get something good (??? O_o ?!):

```
Command Window
>> [mydata.Proprio str2num(cell2mat(eval(tree_full,data)))]

ans =

     0     1
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     1     1
     1     0
     1     1
     1     0
     1     0
     1     1
     1     1
     1     1
     1     1
     1     1
     1     1
```

We can use the function **confusionmat( )** to quickly get the confusion matrix:

```
Command Window
>> confusionmat(mydata.Proprio,str2num(cell2mat(eval(tree_full,data))))

ans =

    11     1
     3     9

fx >>
```

Since MATLAB 2017a the syntax has change quite a bit, we must use the function **fitctree( )**:

```
MATLAB Command Window
» mydata=dataset('File','c:/tmp/CART.csv','Delimiter',';');
» Proprio=logical(mydata.Proprio);
» mydata=mydata(:,1:2);
» model=fitctree(mydata,Proprio,'SplitCriterion','gdi')

model =

  ClassificationTree
             PredictorNames: {'Revenus'  'Surface'}
               ResponseName: 'Y'
    CategoricalPredictors: []
                ClassNames: [0 1]
           ScoreTransform: 'none'
          NumObservations: 24

» |
```

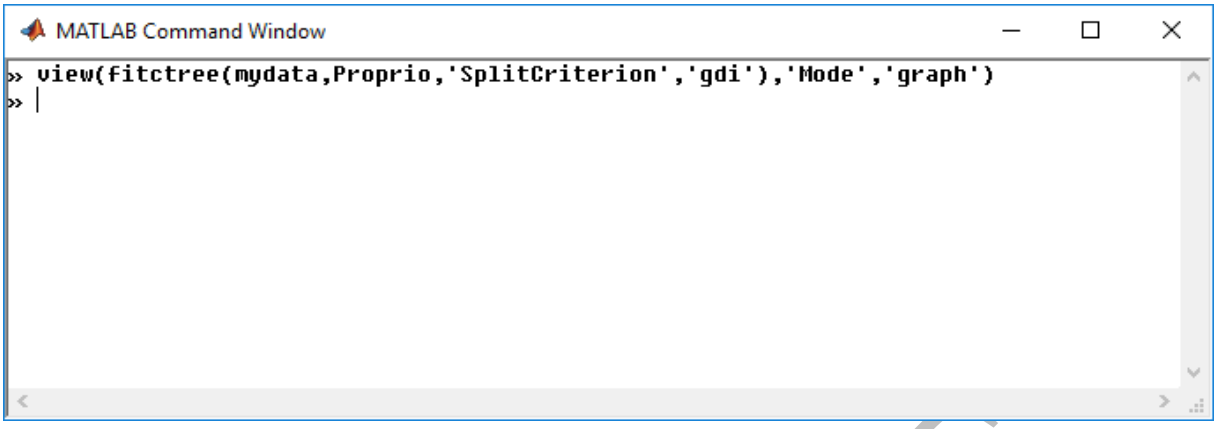
and to get the model details we write:

```
MATLAB Command Window
» view(model)

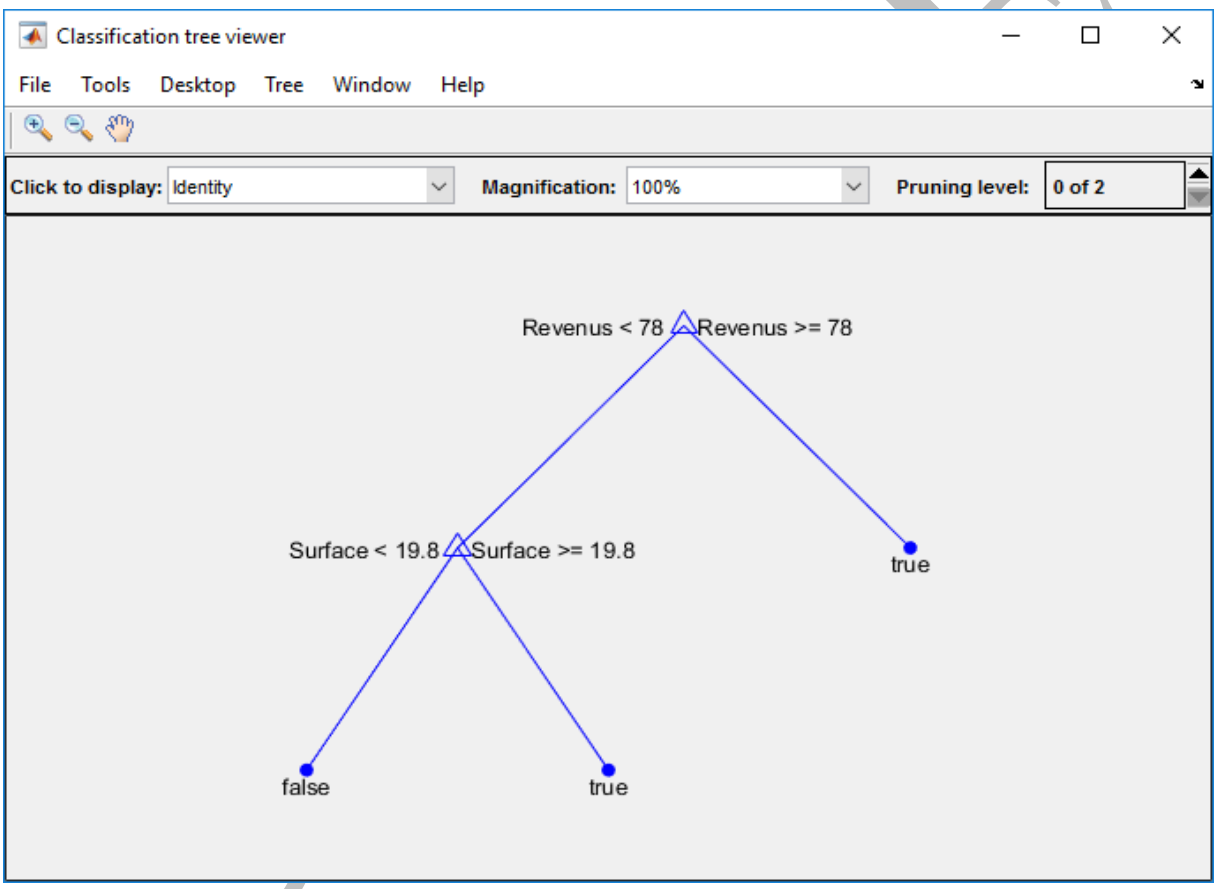
Decision tree for classification
1  if Revenus<78 then node 2 elseif Revenus>=78 then node 3 else false
2  if Surface<19.8 then node 4 elseif Surface>=19.8 then node 5 else false
3  class = true
4  class = false
5  class = true

» |
```

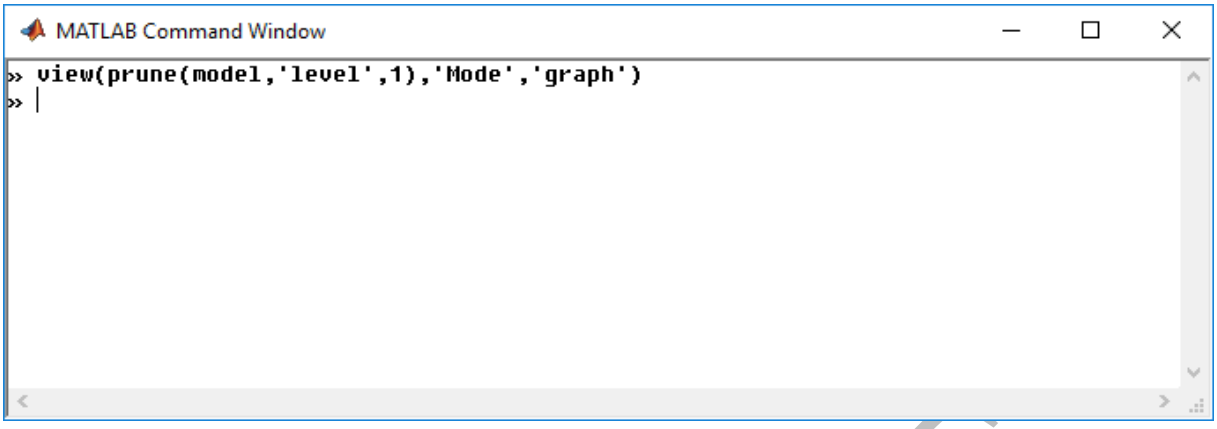
and to visualize, we use the function **view( )** on our model:



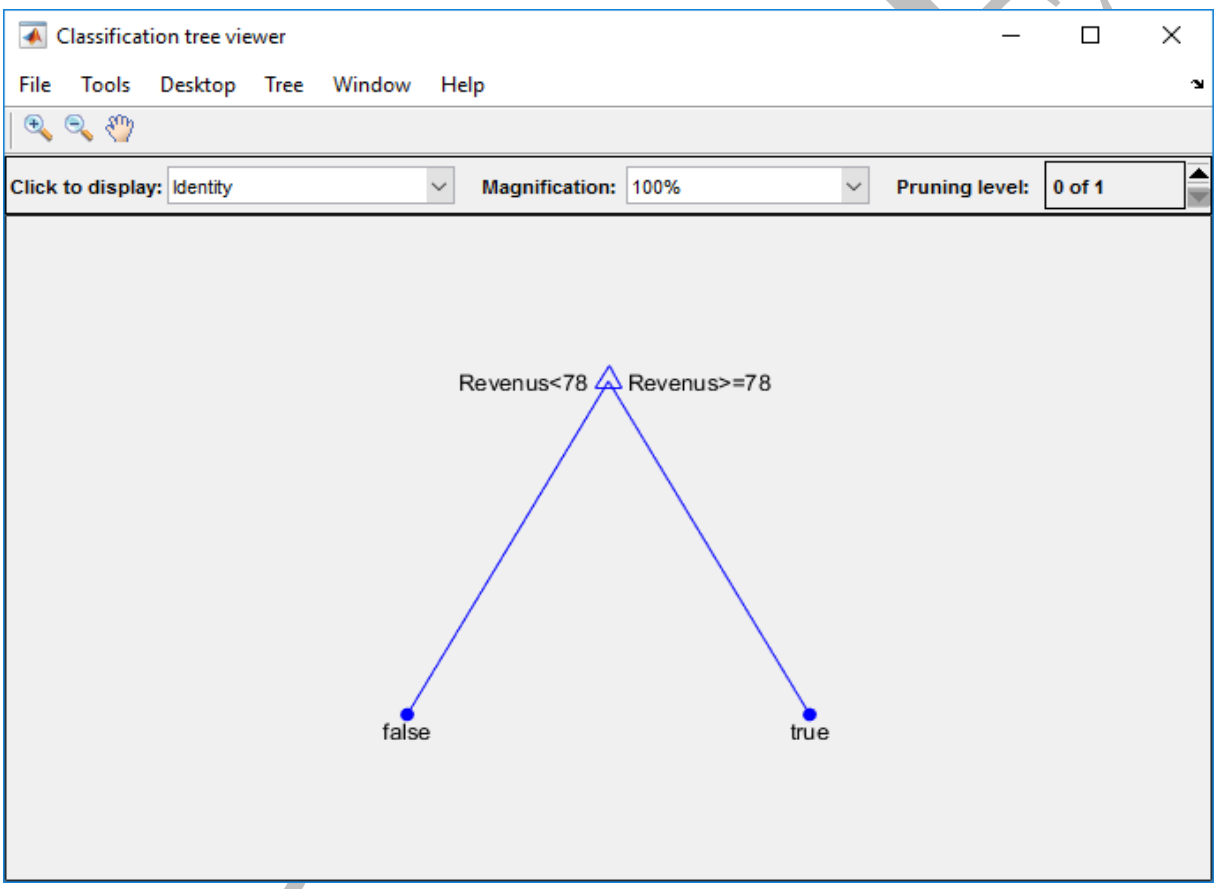
That gives:



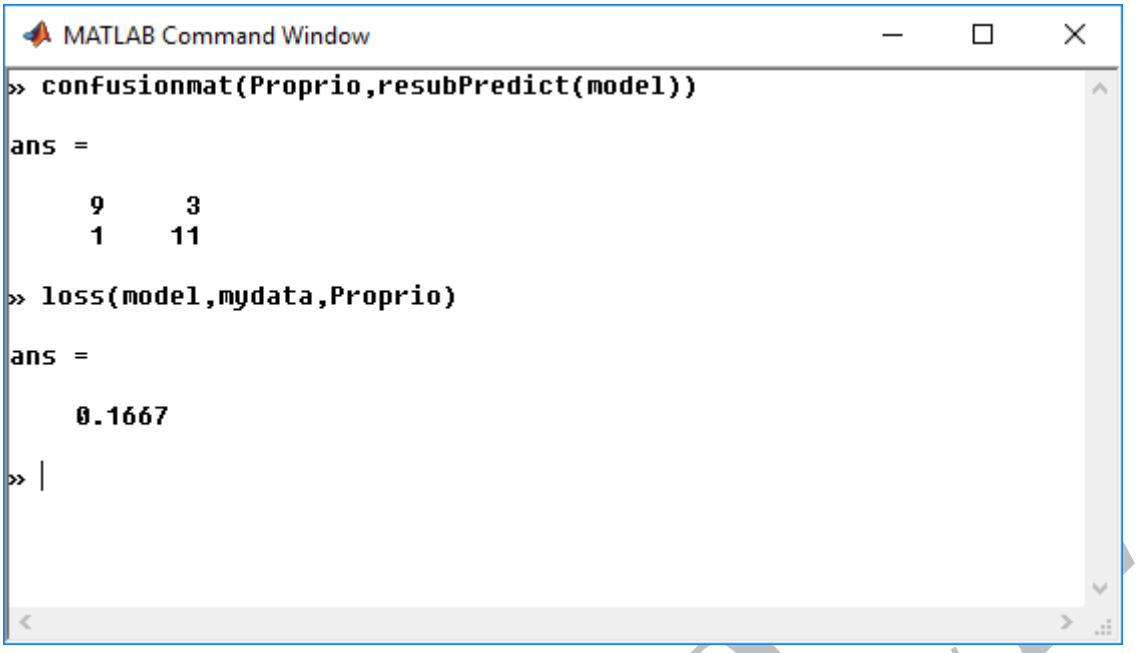
and we can do again some pruning:



to get:



We can get the confusion matrix in a bit easier way:



and alos the loss of the model.

Non classical multidimensional scaling

As always the purpose here is to see if we get the same results as in theoretical course and also compare it to the results obtained with R and Microsoft Excel.

For this purpose we will use the same dissimilarity matrix as in the theoretical course with the function **mdscale( )**:

SAMPLE

C:\Documents and Settings\Administrateur\Mes documents\MultidimensionnalScaling.m

EDITOR PUBLISH VIEW

```
clear all;close all;clc;
countries = ...
{'Brazil','Congo','Cuba','Egypt','France','India','Israel','Japan','Chin
'Russia','USA','Yougoslavie'};
D = [     0  4.17  3.72 5.56 4.28 4.50 5.17 5.50 6.61 5.94 3.61 5.83;
       4.17     0  4.44 4.00 5.00 4.17 5.67 5.61 5.00 5.61 6.61 5.50;
       3.72  4.44     0 3.83 4.89 5.00 5.39 6.06 3.50 3.56 5.83 3.89;
       5.56  4.00  3.83    0 4.22 3.17 4.33 5.17 4.61 4.61 5.67 4.72;
       4.28  5.00  4.89 4.22    0 5.56 5.00 4.78 5.33 3.94 3.06 4.28;
       4.50  4.17  5.00 3.17 5.56    0 4.89 4.50 4.89 4.50 4.72 5.00;
       5.17  5.67  5.39 4.33 5.00 4.89    0 4.17 6.00 4.83 3.06 4.56;
       5.50  5.61  6.06 5.17 4.78 4.50 4.17    0 4.83 4.39 2.94 4.72;
       6.61  5.00  3.50 4.61 5.33 4.89 6.00 4.83    0 3.28 6.44 3.94;
       5.94  5.61  3.56 4.61 3.94 4.50 4.83 4.39 3.28    0 4.00 2.33;
       3.61  6.61  5.83 5.67 3.06 4.72 3.06 2.94 6.44 4.00    0 5.44;
       5.83  5.50  3.89 4.72 4.28 5.00 4.56 4.72 3.94 2.33 5.44    0];
Y = mdscale(D,2,'Criterion','metricstress')
plot(Y(:,1),Y(:,2),'o');
grid on;
text(Y(:,1),Y(:,2),countries);
```

And we get:

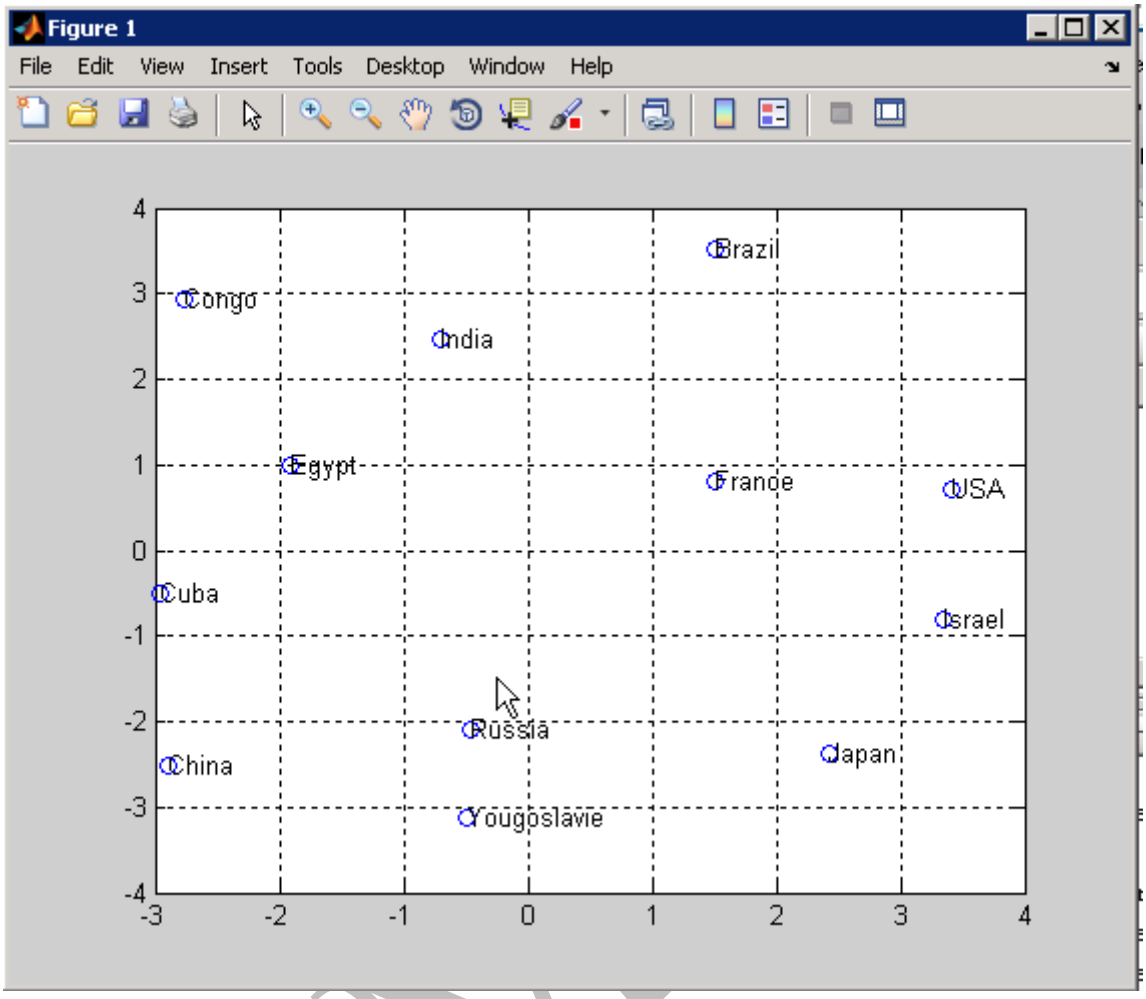
Command Window

```
Y =

    1.5019    3.5289
   -2.7597    2.9266
   -2.9667   -0.4965
   -1.9183    0.9899
    1.5098    0.7958
   -0.6989    2.4630
    3.3481   -0.8143
    2.4254   -2.3751
   -2.8981   -2.5167
   -0.4575   -2.0914
    3.4077    0.7098
   -0.4936   -3.1200

fx >>
```

And:

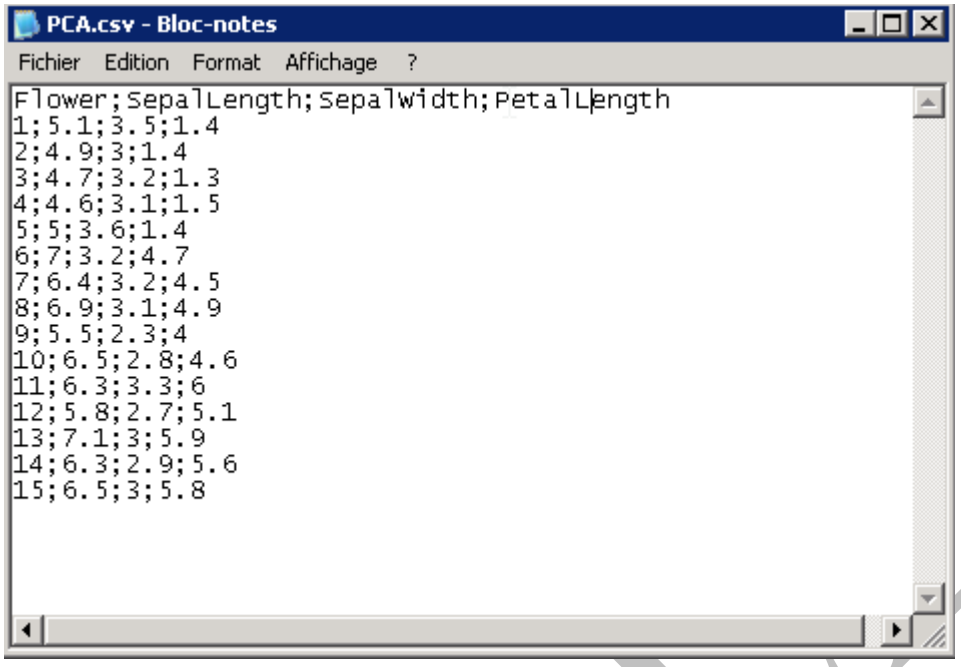


Thus the results are near for the third method used in the R training and far away from the calculations made with Microsoft Excel. But the interpretation of clusters still remains the same.

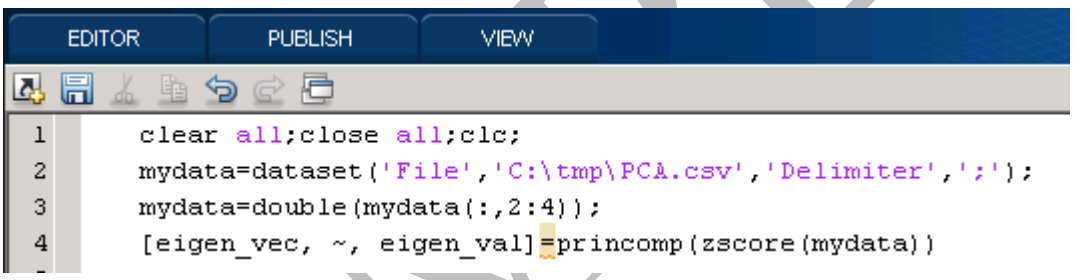
Parametric Principal Component Analysis

As we saw it in the theoretical statistic course, PCA is a very important tool in a lot of different domains. The proof of the parametric model was very long but the results powerful.

In the MATLAB exercise we will as always use exactly the same data set as the one unsed in the theoretical course. That is: a subset of the Fisher Iris set:

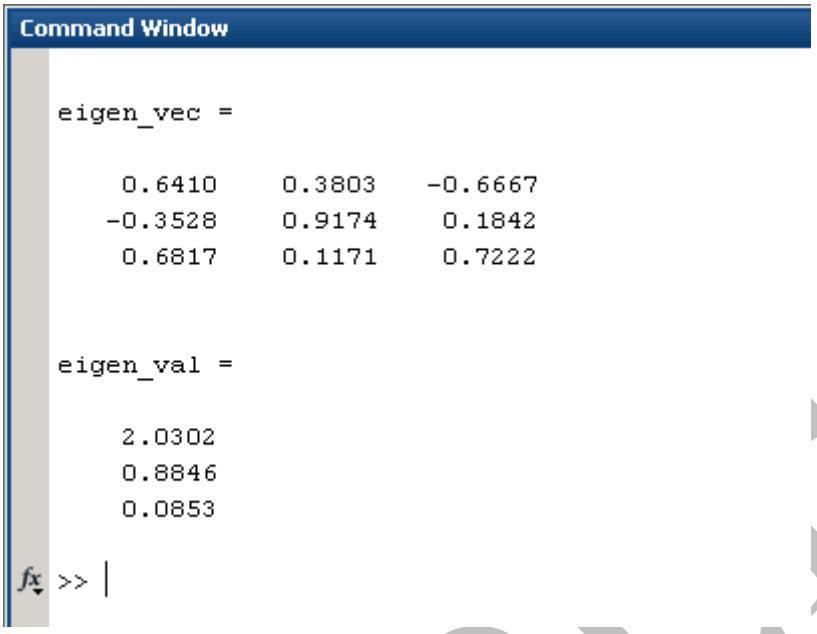


After we just need to import the data and use the function **princomp( )**:

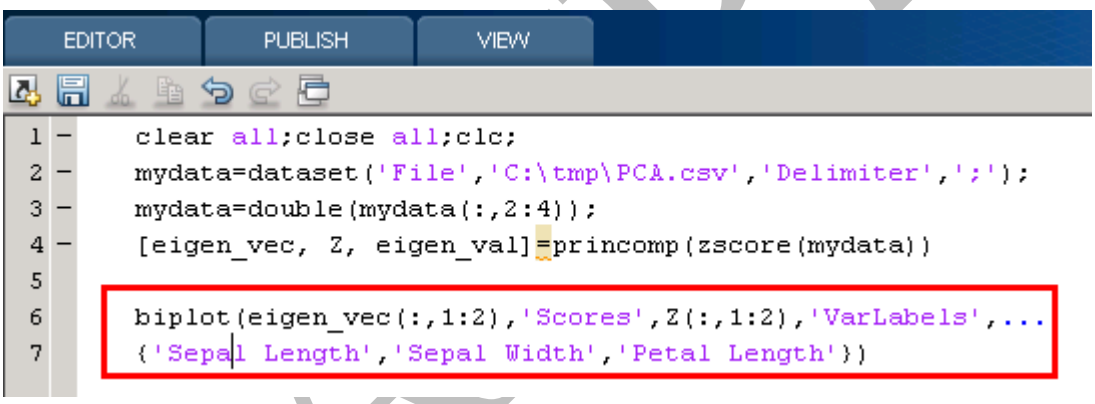


SAMPLE

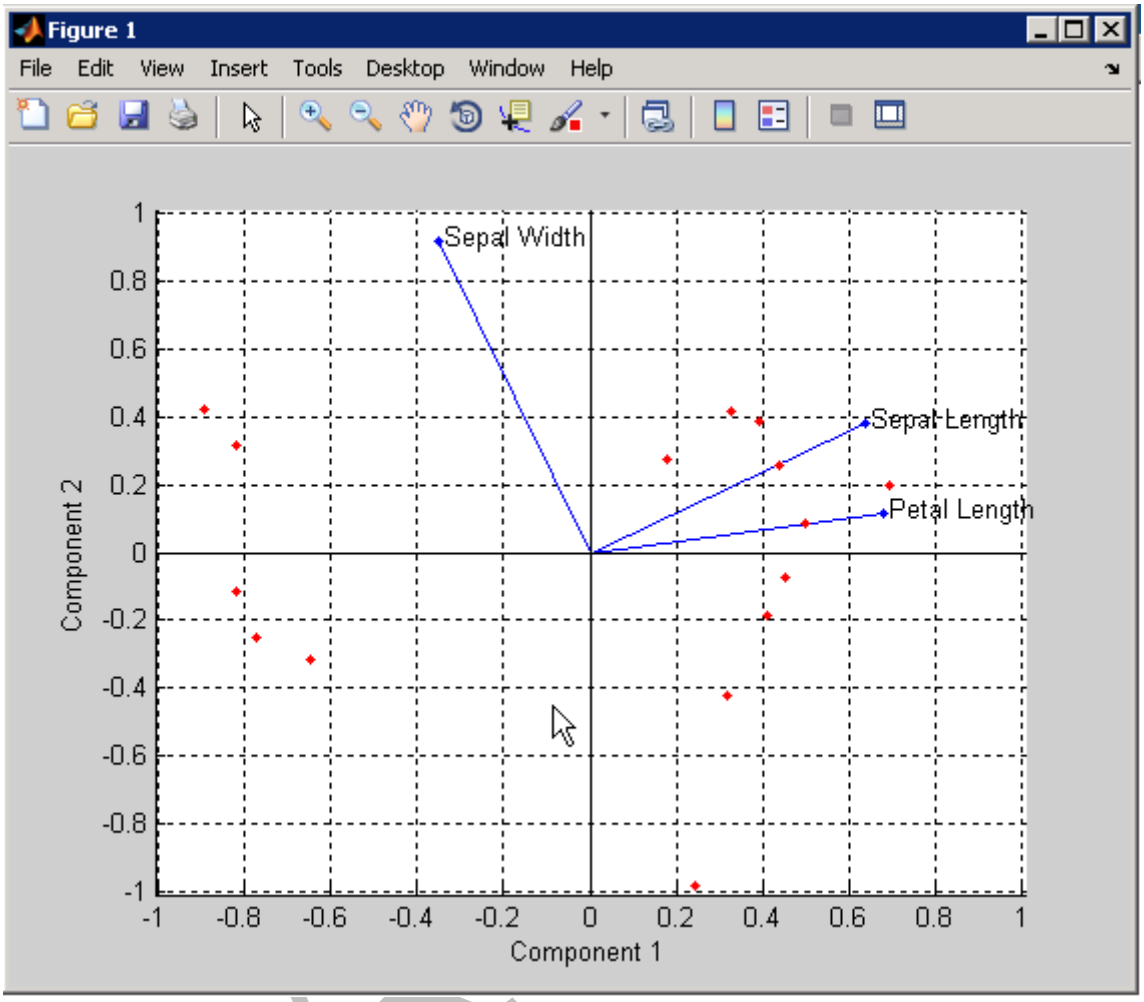
We get exactly the same eigen vectors and eigen values as the calculations made by hand:



We can also plot the famous biplot using the **biplot( )** function:



We get the following factorial plan that has the same scatter point structure as in the theoretical course. But the values of the scatter points are not the same as the one of Minitab and R or even as the one calculated by hand but the vectors are corresponding to the one we calculated by hand (but not corresponding with Minitab and R):

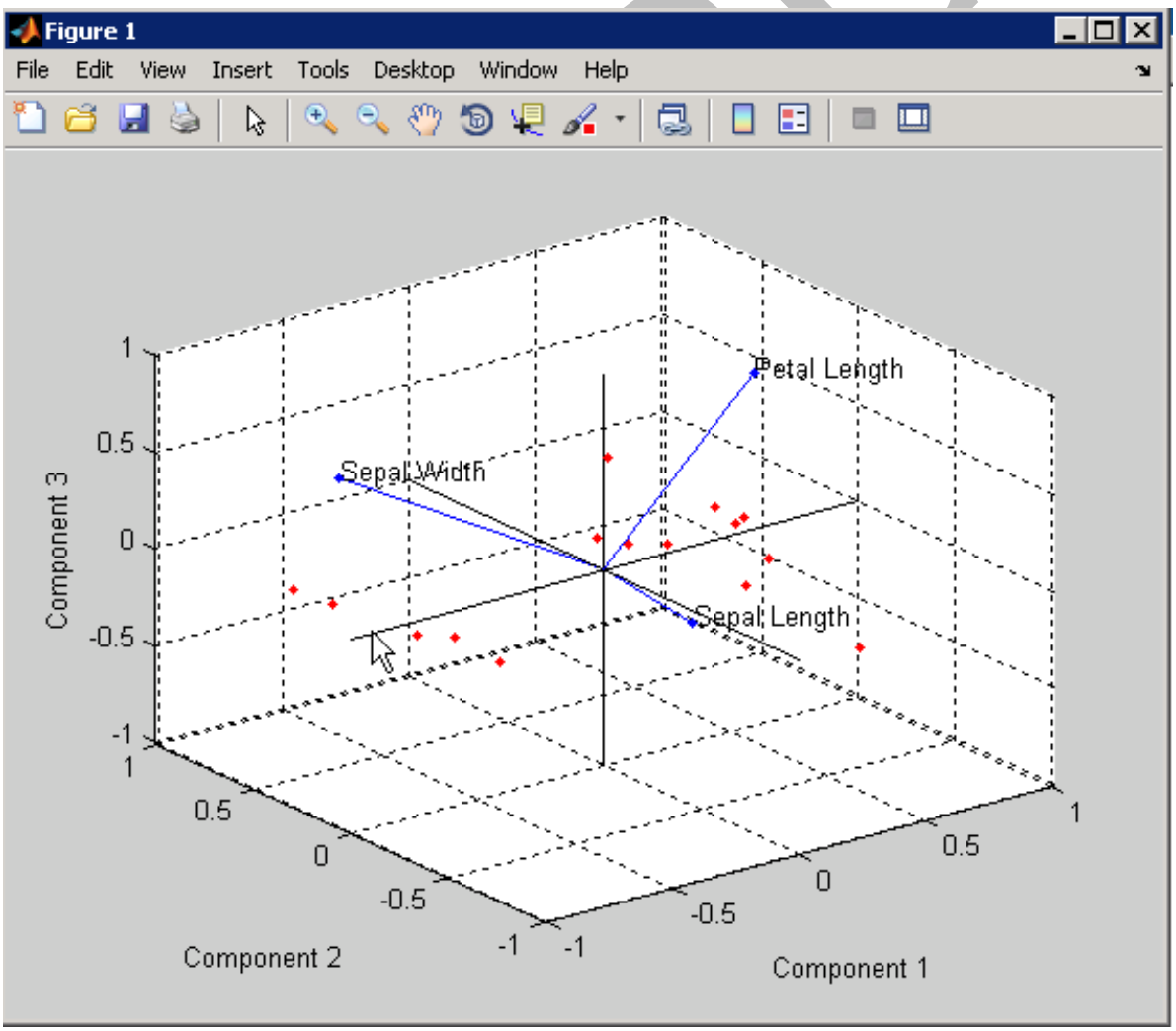


SA

And if you we want all the stuff in 3D:

```
1 -   clear all;close all;clc;
2 -   mydata=dataset('File','C:\tmp\PCA.csv','Delimiter',';');
3 -   mydata=double(mydata(:,2:4));
4 -   [eigen_vec, Z, eigen_val]=princomp(zscore(mydata))
5
6 -   biplot(eigen_vec(:,1:2),'Scores',Z(:,1:2),'VarLabels',...
7     {'Sepal Length','Sepal Width','Petal Length'})
8
9 -   biplot(eigen_vec(:,1:3),'Scores',Z(:,1:3),'VarLabels',...
10    {'Sepal Length','Sepal Width','Petal Length'})
```

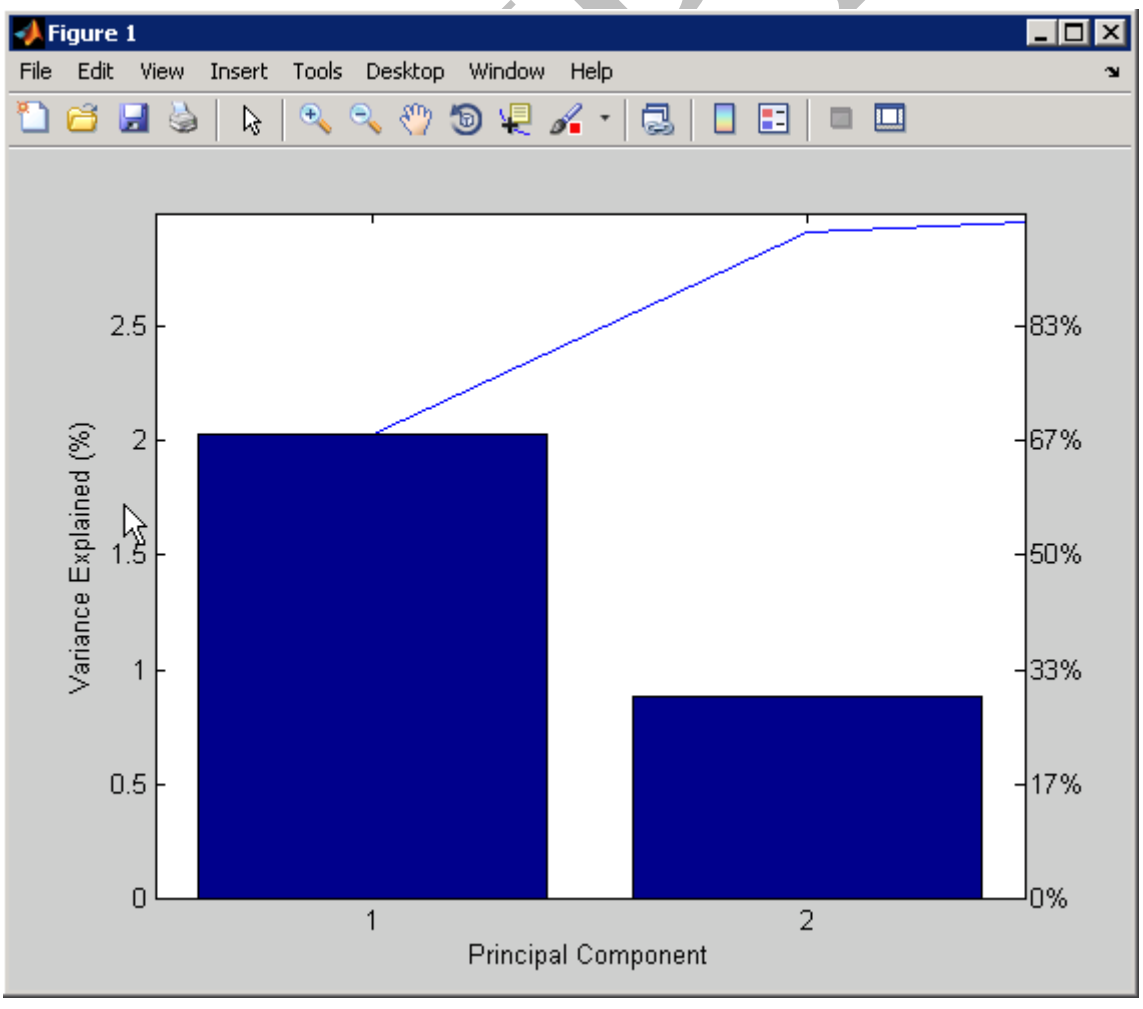
We get:



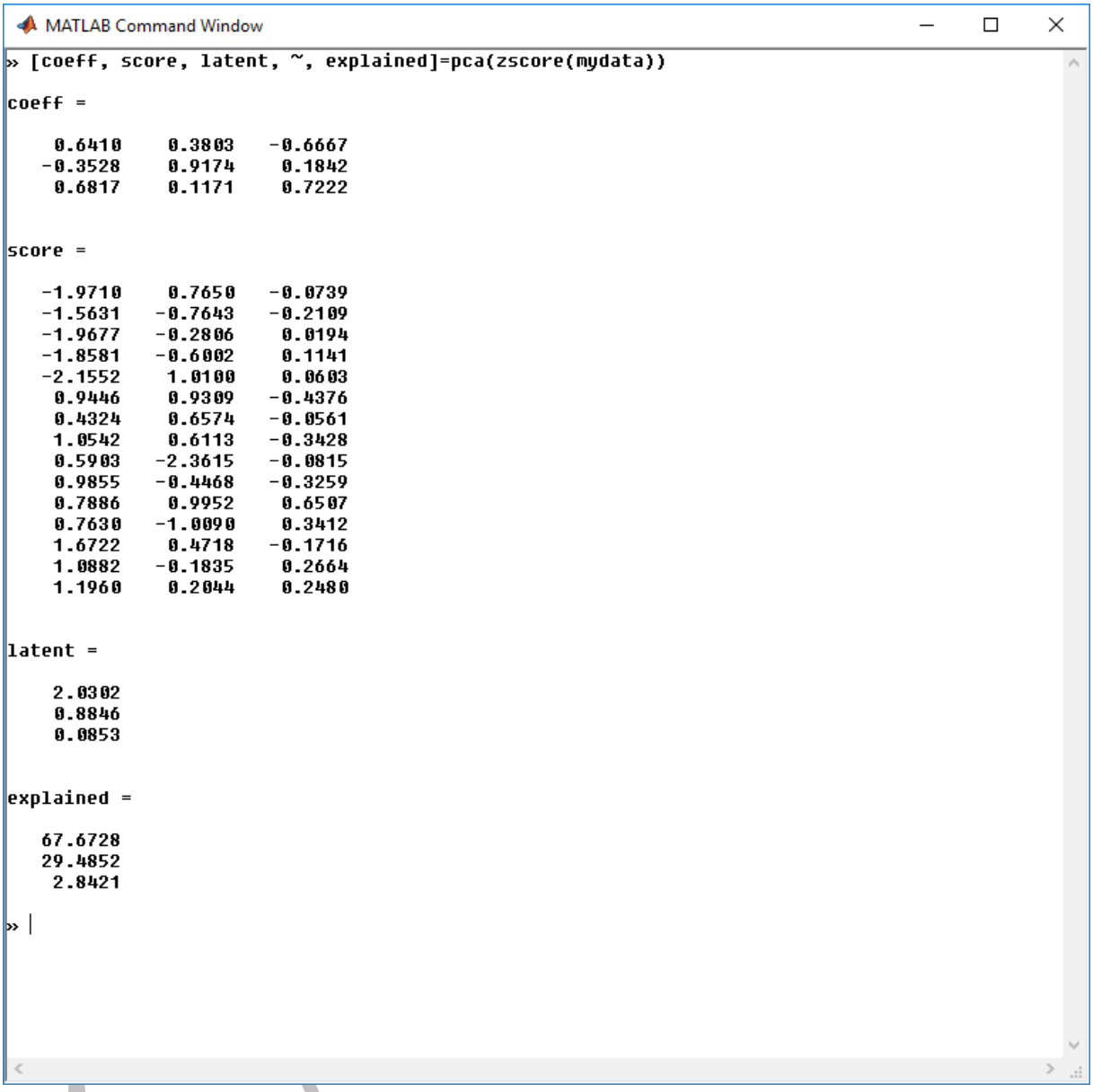
We can also add the famous Pareto eigenvalues ponderation:

```
1  -    clear all;close all;clc;
2  -    mydata=dataset('File','C:\tmp\PCA.csv','Delimiter',';');
3  -    mydata=double(mydata(:,2:4));
4  -    [eigen_vec, Z, eigen_val]=princomp(zscore(mydata))
5
6  -    biplot(eigen_vec(:,1:2),'Scores',Z(:,1:2),'VarLabels',...
7       {'Sepal Length','Sepal Width','Petal Length'})
8
9  -    biplot(eigen_vec(:,1:3),'Scores',Z(:,1:3),'VarLabels',...
10      {'Sepal Length','Sepal Width','Petal Length'})
11
12 -    pareto(eigen_val);
13 -    xlabel('Principal Component');
14 -    ylabel('Variance Explained (%)');
```

So we get the following "scree plot":



Since MATLAB 2017a the syntax has change quite a bit. We have to use the function **pca( )**:



Linear Discriminant Analysis (Fisher Discriminant)

The purpose here will be to see how we can get with MATLAB the LDA result the we have proved in the data mining theoretical course using the Eigenvector Projection method (at the opposite of the Likelihood LDA method).

As far as I know there is no LDA integrated analysis in MATLAB 2013a using this method. This is why we will have to calculate everything ourself.

We type in our small data set values and make also the small mean, covariance and within class scatter matrix calculations:

```
EDITOR  PUBLISH  VIEW

1 -  clear all;close all;clc
2 -  X1=[4,2;
3        2,4;
4        2,3;
5        3,6;
6        4,4];%samples for class 1
7 -  X2=[9,10;
8        6,8;
9        9,5;
10       8,7;
11       10,8];%samples for class 2
12   %class means
13 -  mu1=mean(X1)
14 -  mu2=mean(X2)
15 -  S1=cov(X1)
16 -  S2=cov(X2)
17   %within class scatter matrix
18 -  Sw=S1+S2
```

This give us (see next page):

```
Command Window

mu1 =

    3.0000    3.8000


mu2 =

    8.4000    7.6000


S1 =

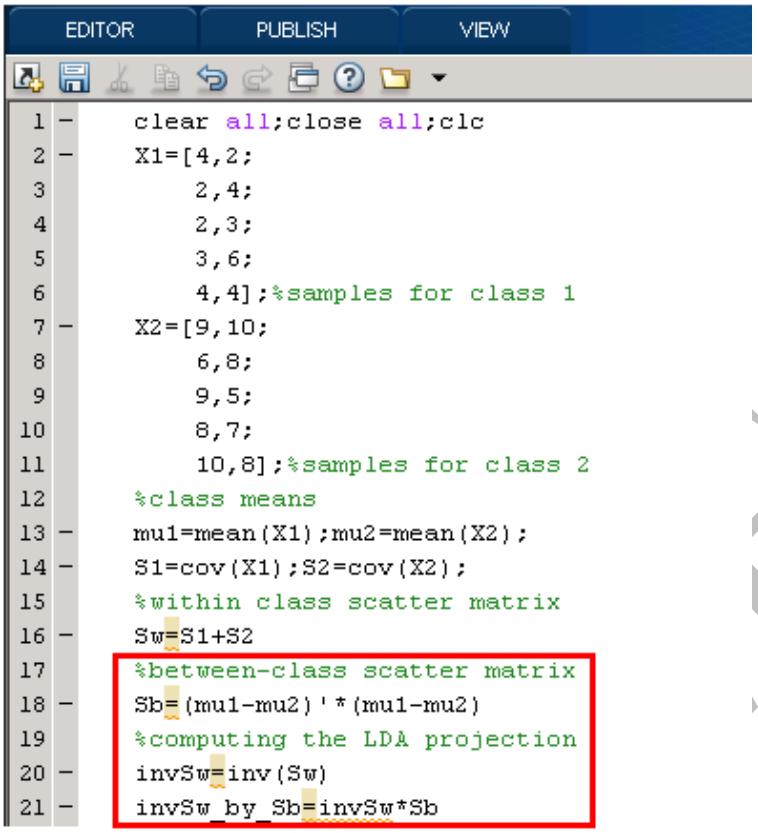
    1.0000   -0.2500
   -0.2500    2.2000


S2 =

    2.3000   -0.0500
   -0.0500    3.3000

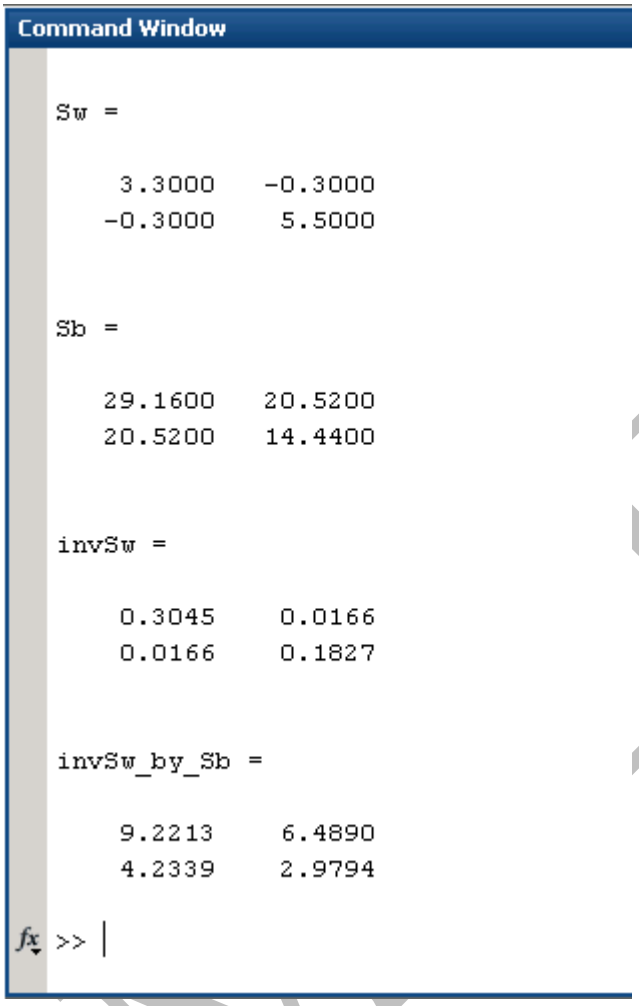
fx >>
```

This are the same results as those calculated by hand. After we add the part to calculate the LDA projection:



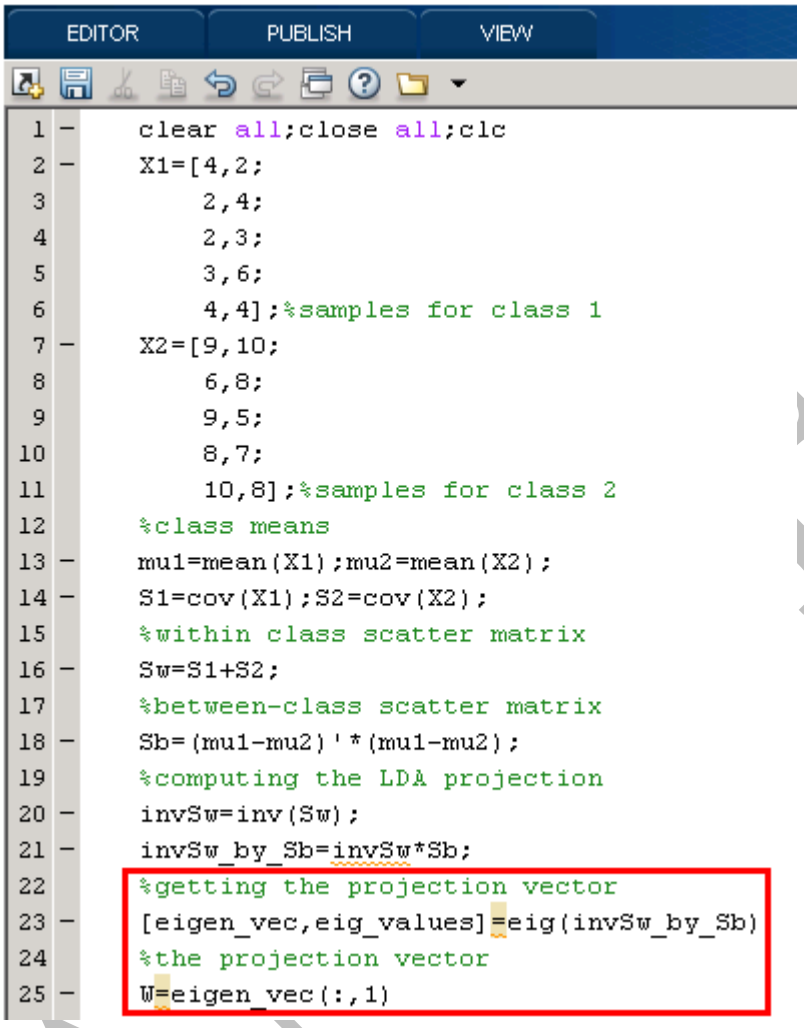
This give us also the same results as those calculated by hand in the theoretical course:

SAMPLE


Command Window
Sw =
3.3000 -0.3000
-0.3000 5.5000
Sb =
29.1600 20.5200
20.5200 14.4400
invSw =
0.3045 0.0166
0.0166 0.1827
invSw_by_Sb =
9.2213 6.4890
4.2339 2.9794
>>

SAMPLE

And to finish we calculate the eigenvectors and eigenvalues and also the projection vector:




```
clear all;close all;clc
X1=[4,2;
    2,4;
    2,3;
    3,6;
    4,4];%samples for class 1
X2=[9,10;
    6,8;
    9,5;
    8,7;
    10,8];%samples for class 2
%class means
mu1=mean(X1);mu2=mean(X2);
S1=cov(X1);S2=cov(X2);
%within class scatter matrix
Sw=S1+S2;
%between-class scatter matrix
Sb=(mu1-mu2)'*(mu1-mu2);
%computing the LDA projection
invSw=inv(Sw);
invSw_by_Sb=invSw*Sb;
%getting the projection vector
[eigen_vec,eig_values]=eig(invSw_by_Sb)
%the projection vector
W=eigen_vec(:,1)
```

This give us also the same results as those calculated by hand:

```
Command Window

eigen_vec =

    0.9088   -0.5755
    0.4173    0.8178


eig_values =

   12.2007         0
         0    0.0000


W =

    0.9088
    0.4173

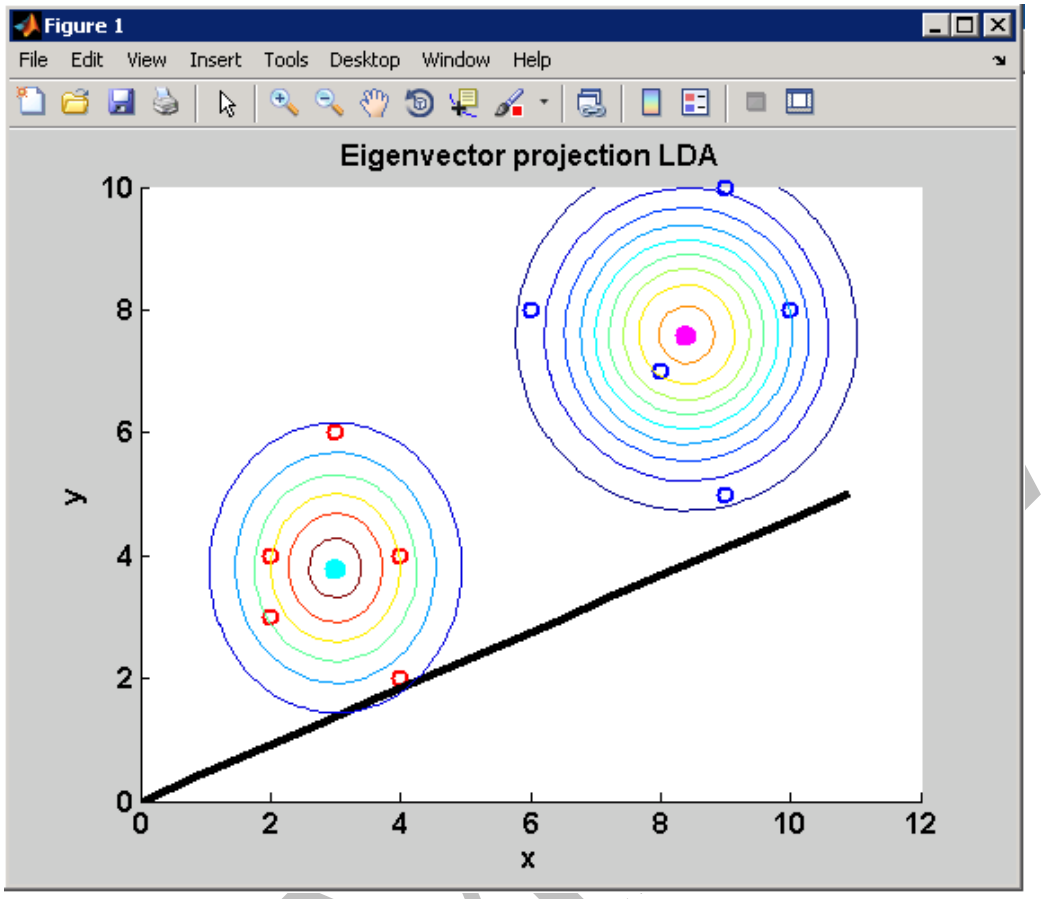
fx >>
```

Finally we just need to plot:

```
EDITOR  PUBLISH  VIEW

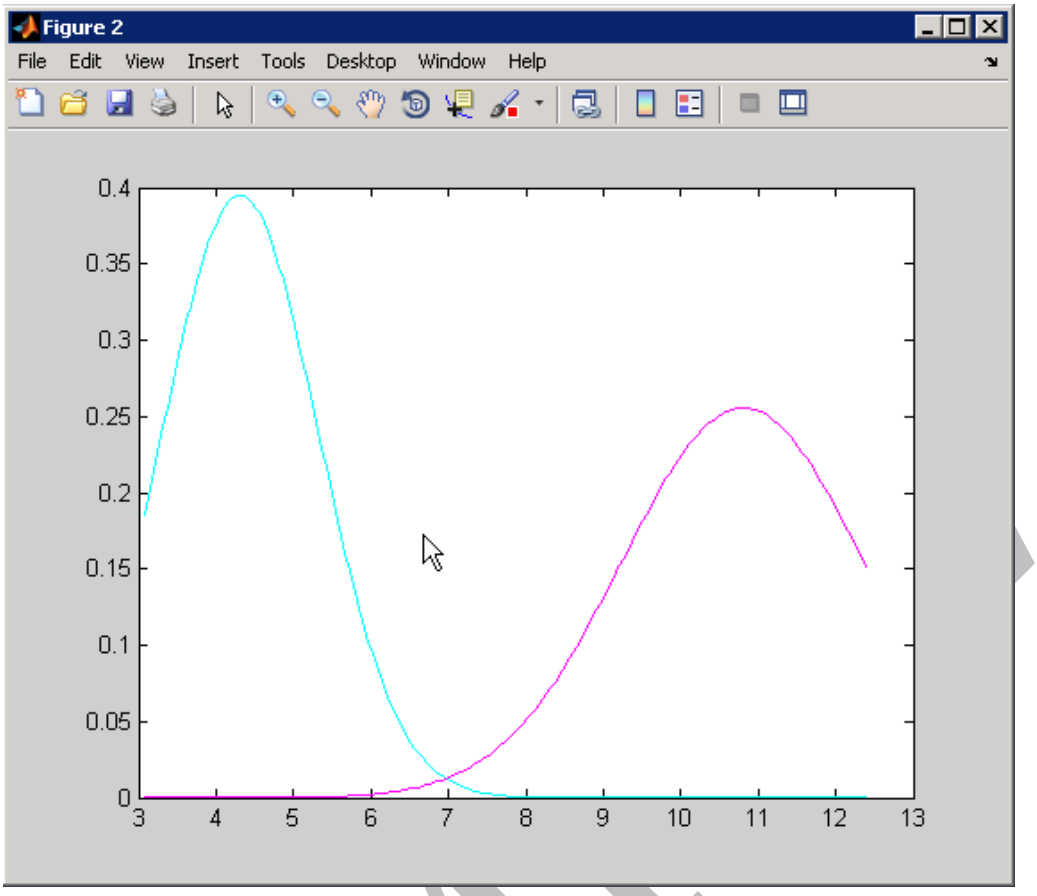
26      %we plot the stuff
27 -    hfig=figure;
28 -    X1=X1';X2=X2';
29 -    axes1=axes('Parent',hfig','Fontweight','bold','FontSize',12);
30 -    hold('all');
31 -    xlabel('X_1');ylabel('X_2');
32
33 -    scatter(X1(1,:),X1(2,:),'r','LineWidth',2,'Parent',axes1);
34 -    scatter(X2(1,:),X2(2,:),'b','LineWidth',2,'Parent',axes1);
35 -    plot(mean(X1(1,:)),mean(X1(2,:)),'co','MarkerSize',8,'MarkerEdgeColor','c',...
36          'Color','c','LineWidth',2,'MarkerFaceColor','c')
37 -    plot(mean(X2(1,:)),mean(X2(2,:)),'mo','MarkerSize',8,'MarkerEdgeColor','m',...
38          'Color','m','LineWidth',2,'MarkerFaceColor','m')
39 -    t=0:12;
40 -    line_x=t.*W(1);
41 -    line_y=t.*W(2);
42 -    plot(line_x,line_y,'k-','LineWidth',3);
43 -    ezcontour(@(x,y) mvnpdf([x y], mu1, [std(X1(1,:)) 0;0 std(X1(2,:))]), xlim(), ylim())
44 -    ezcontour(@(x,y) mvnpdf([x y], mu2, [std(X2(1,:)) 0;0 std(X2(2,:))]), xlim(), ylim())
45      title('Eigenvector projection LDA');
```

We get the famous LDA Gaussian density projection chart:



Or using another famous other representation we have:

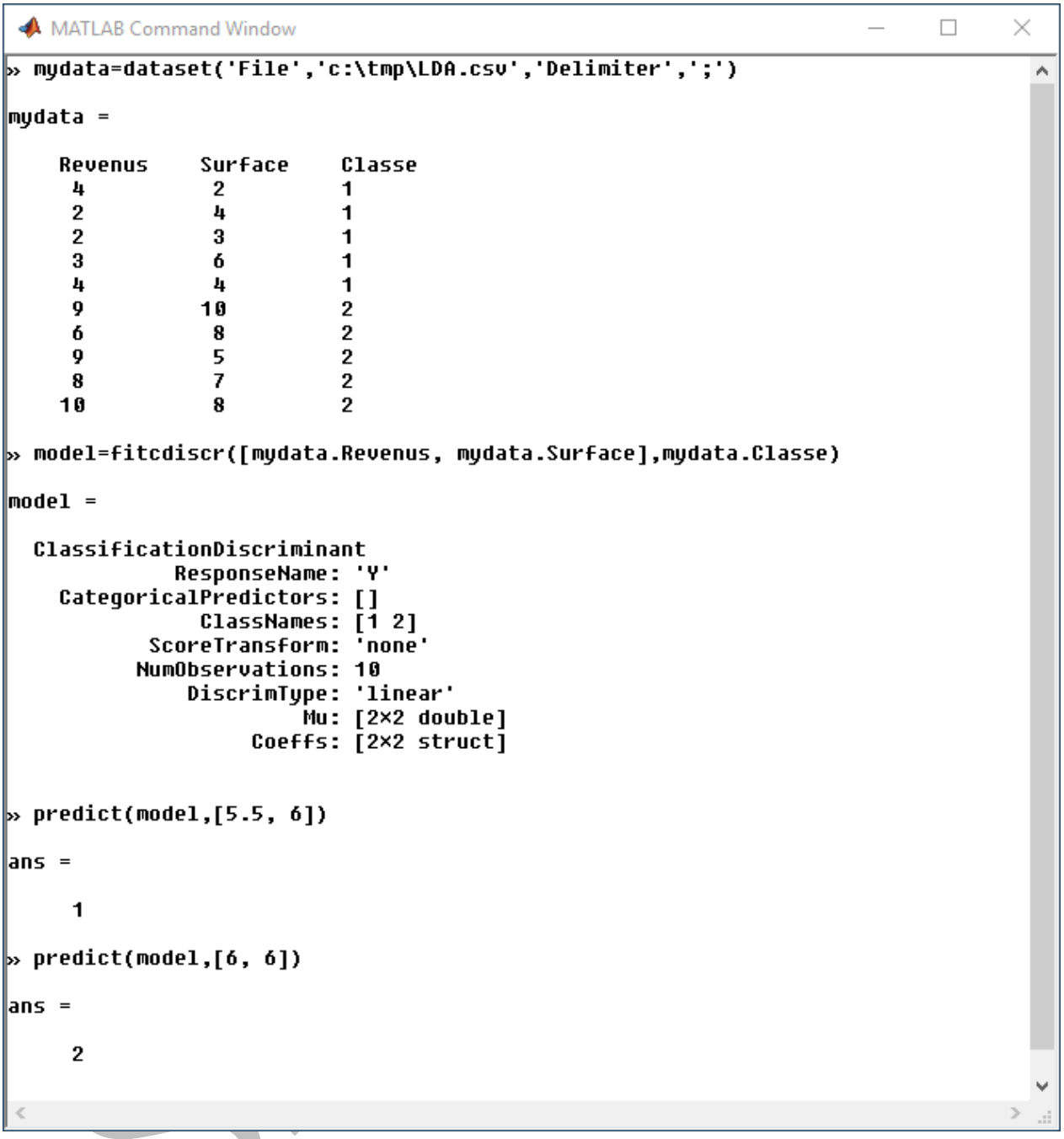
```
46
47      %project data samples along the projection axes
48 -    y1_w=W'*X1;
49 -    y2_w=W'*X2;
50 -    minY=min([min(y1_w),min(y2_w)]);
51 -    maxY=max([max(y1_w),max(y2_w)]);
52 -    y_w=minY:0.05:maxY;
53
54 -    y1_w_mu=mean(y1_w);
55 -    y2_w_mu=mean(y2_w);
56 -    y1_w_sigma=std(y1_w);
57 -    y2_w_sigma=std(y2_w);
58
59 -    figure
60 -    plot(y_w',normpdf(y_w',y1_w_mu,y1_w_sigma),'c');
61 -    hold on;
62 -    plot(y_w',normpdf(y_w',y2_w_mu,y2_w_sigma),'m');
```



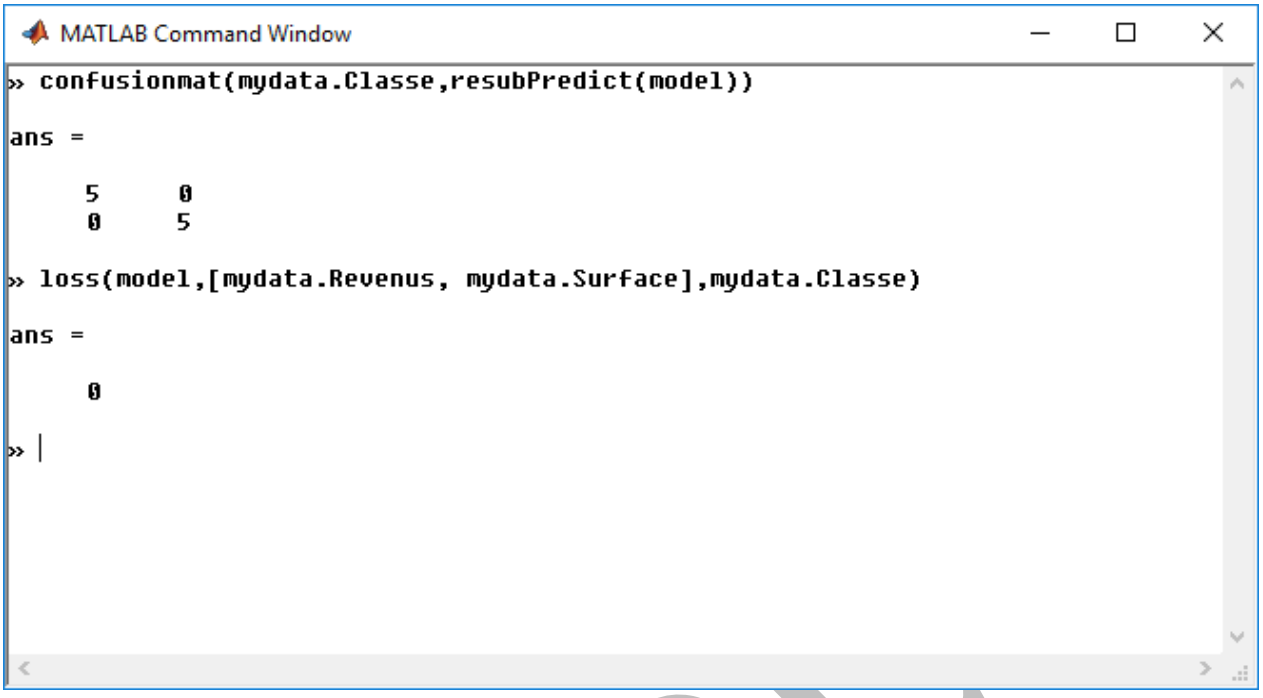
Everything corresponds for sure to calculation made by hand in the theoretical course.

At least since MATLAB 2017a the system has been automated. We can use a function named **fitcdiscr( )**:

SAMPLE



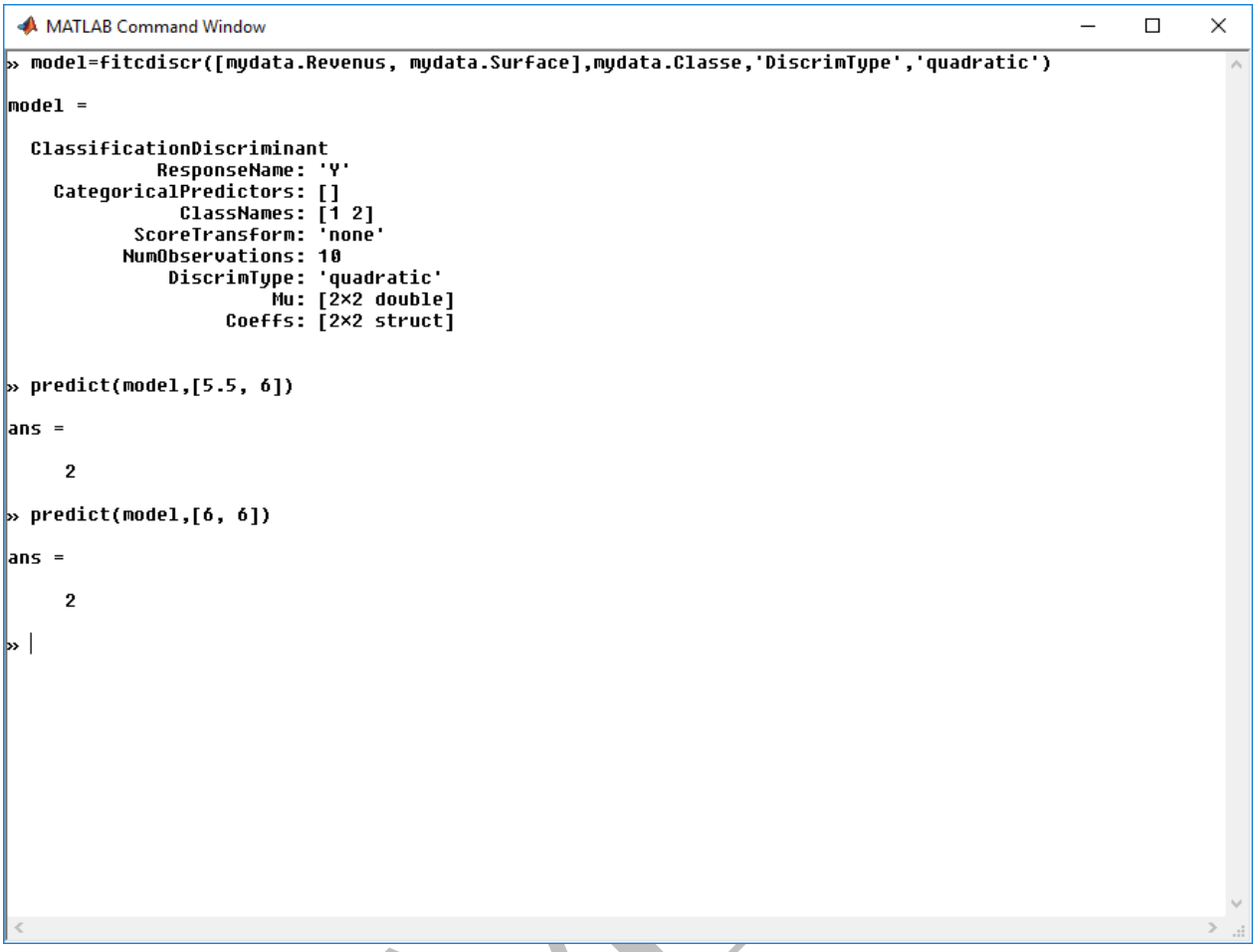
And as always we can calculate the confusion matrix and the loss of the model:



Quadratic Discriminant Analysis

Since at least MATLAB 2017a the quadratic discriminant analysis (QDA) has also been implemented. Still with function **fitcdiscr( )**:

SAMPLE

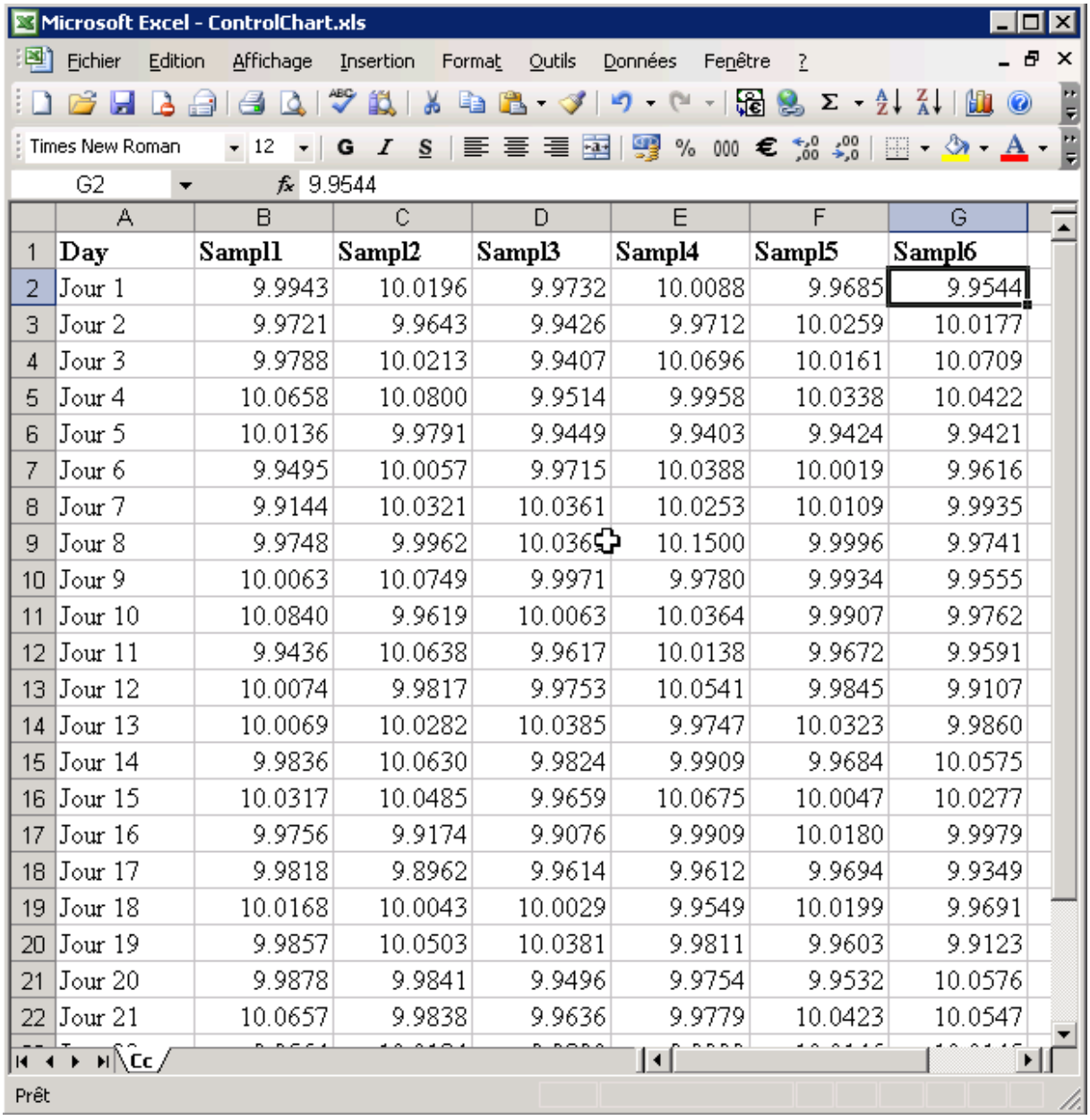

MATLAB Command Window
» model=fitcdiscr([mydata.Revenus, mydata.Surface],mydata.Classe,'DiscrimType','quadratic')
model =
ClassificationDiscriminant
ResponseName: 'Y'
CategoricalPredictors: []
ClassNames: [1 2]
ScoreTransform: 'none'
NumObservations: 10
DiscrimType: 'quadratic'
Mu: [2×2 double]
Coeffs: [2×2 struct]
» predict(model,[5.5, 6])
ans =
2
» predict(model,[6, 6])
ans =
2
»

SAMPLE

Control charts (CC)

This is a big and important subject in quantitative quality control. MATLAB is not a flexible as Minitab but has at least the (very) minimum we can expect of an engineering software with a Statistical Toolbox.

To study the continuous control chart we will use the same data as in the theoretical course. This is to say:



	A	B	C	D	E	F	G
1	**Day**	**Sampl1**	**Sampl2**	**Sampl3**	**Sampl4**	**Sampl5**	**Sampl6**
2	Jour 1	9.9943	10.0196	9.9732	10.0088	9.9685	9.9544
3	Jour 2	9.9721	9.9643	9.9426	9.9712	10.0259	10.0177
4	Jour 3	9.9788	10.0213	9.9407	10.0696	10.0161	10.0709
5	Jour 4	10.0658	10.0800	9.9514	9.9958	10.0338	10.0422
6	Jour 5	10.0136	9.9791	9.9449	9.9403	9.9424	9.9421
7	Jour 6	9.9495	10.0057	9.9715	10.0388	10.0019	9.9616
8	Jour 7	9.9144	10.0321	10.0361	10.0253	10.0109	9.9935
9	Jour 8	9.9748	9.9962	10.036	10.1500	9.9996	9.9741
10	Jour 9	10.0063	10.0749	9.9971	9.9780	9.9934	9.9555
11	Jour 10	10.0840	9.9619	10.0063	10.0364	9.9907	9.9762
12	Jour 11	9.9436	10.0638	9.9617	10.0138	9.9672	9.9591
13	Jour 12	10.0074	9.9817	9.9753	10.0541	9.9845	9.9107
14	Jour 13	10.0069	10.0282	10.0385	9.9747	10.0323	9.9860
15	Jour 14	9.9836	10.0630	9.9824	9.9909	9.9684	10.0575
16	Jour 15	10.0317	10.0485	9.9659	10.0675	10.0047	10.0277
17	Jour 16	9.9756	9.9174	9.9076	9.9909	10.0180	9.9979
18	Jour 17	9.9818	9.8962	9.9614	9.9612	9.9694	9.9349
19	Jour 18	10.0168	10.0043	10.0029	9.9549	10.0199	9.9691
20	Jour 19	9.9857	10.0503	10.0381	9.9811	9.9603	9.9123
21	Jour 20	9.9878	9.9841	9.9496	9.9754	9.9532	10.0576
22	Jour 21	10.0657	9.9838	9.9636	9.9779	10.0423	10.0547

We import the data as usual in MATLAB:

Command Window

```
>> mydata=dataset('XLSFile','c:\tmp\ControlChart.xls')

mydata =

    Day              Sampl1    Sampl2    Sampl3    Sampl4    Sampl5    Sampl6
    'Jour 1'         9.9943     10.02    9.9732    10.009    9.9685    9.9544
    'Jour 2'         9.9721    9.9643    9.9426    9.9712    10.026    10.018
    'Jour 3'         9.9788    10.021    9.9407     10.07    10.016    10.071
    'Jour 4'         10.066     10.08    9.9514    9.9958    10.034    10.042
    'Jour 5'         10.014    9.9791    9.9449    9.9403    9.9424    9.9421
    'Jour 6'         9.9495    10.006    9.9715    10.039    10.002    9.9616
    'Jour 7'         9.9144    10.032    10.036    10.025    10.011    9.9935
    'Jour 8'         9.9748    9.9962    10.037     10.15    9.9996    9.9741
    'Jour 9'         10.006    10.075    9.9971     9.978    9.9934    9.9555
    'Jour 10'        10.084    9.9619    10.006    10.036    9.9907    9.9762
    'Jour 11'        9.9436    10.064    9.9617    10.014    9.9672    9.9591
    'Jour 12'        10.007    9.9817    9.9753    10.054    9.9845    9.9107
    'Jour 13'        10.007    10.028    10.039    9.9747    10.032     9.986
    'Jour 14'        9.9836    10.063    9.9824    9.9909    9.9684    10.057
    'Jour 15'        10.032    10.049    9.9659    10.068    10.005    10.028
    'Jour 16'        9.9756    9.9174    9.9076    9.9909    10.018    9.9979
    'Jour 17'        9.9818    9.8962    9.9614    9.9612    9.9694    9.9349
    'Jour 18'        10.017    10.004    10.003    9.9549     10.02    9.9691
    'Jour 19'        9.9857     10.05    10.038    9.9811    9.9603    9.9123
    'Jour 20'        9.9878    9.9841    9.9496    9.9754    9.9532    10.058
    'Jour 21'        10.066    9.9838    9.9636    9.9779    10.042    10.055
    'Jour 22'        9.9564    10.018     9.989    9.9999    10.015    10.015
    'Jour 23'        10.061    9.9443    9.9899    9.9833    9.9947    10.003
```

SAMPLE

P attribute CC

The proportion control chart is one of the most easy to understand CC in the theoretical point of view (and also to read). We will also here use the same data as in the theoretical course to check if we get the same results:

Command Window

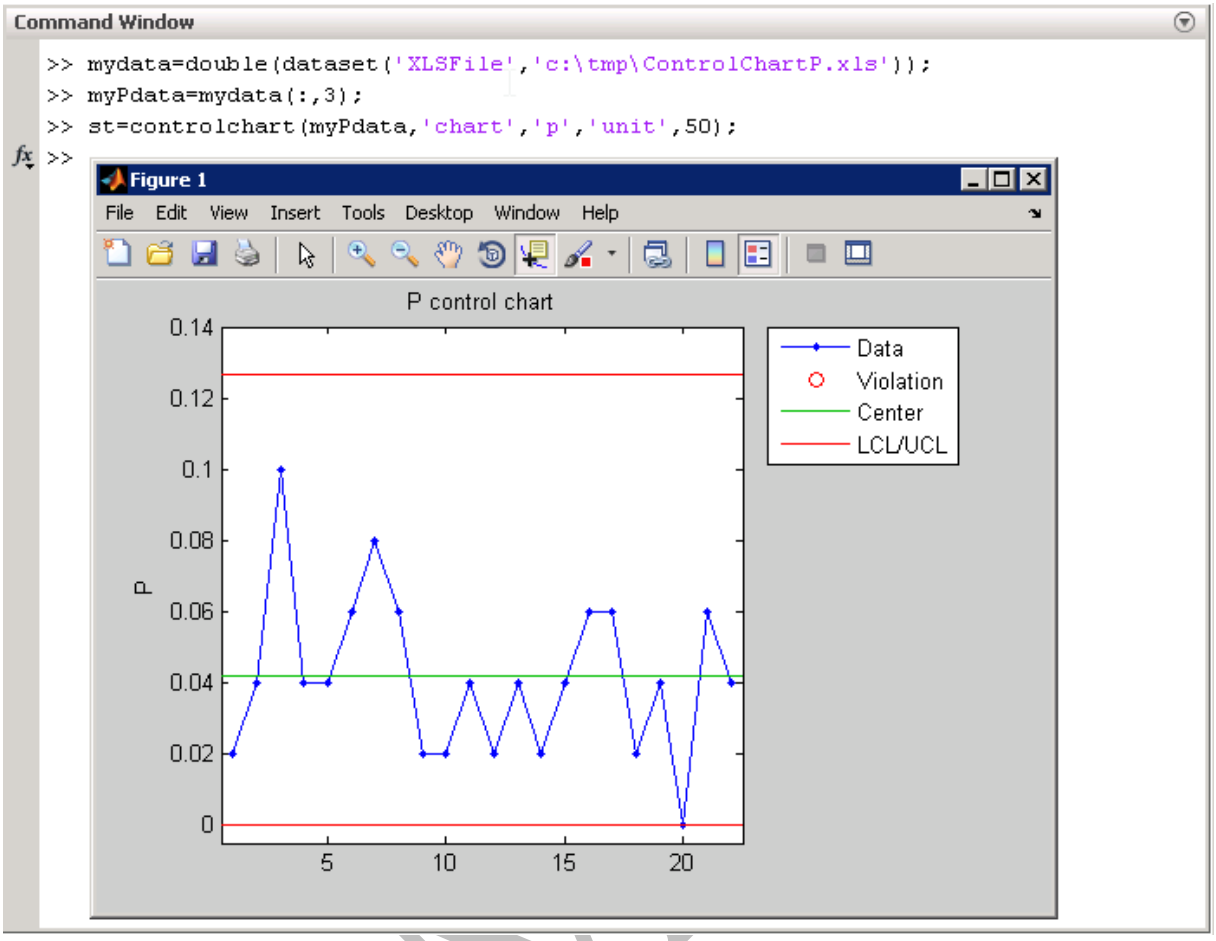
```
>> mydata=dataset('XLSFile','c:\tmp\ControlChartP.xls')

mydata =

    N_Lot    TailleLot    NonConformes
     1        50           1
     2        50           2
     3        50           5
     4        50           2
     5        50           2
     6        50           3
     7        50           4
     8        50           3
     9        50           1
    10        50           1
    11        50           2
    12        50           1
    13        50           2
    14        50           1
    15        50           2
    16        50           3
    17        50           3
    18        50           1
    19        50           2
    20        50           0
    21        50           3
    22        50           2
```

SA

Using once again (...) the function **controlchart( )** we get:



Here also we get the same values as in the theoretical course except for the first three points.

SAM

NP attribute CC

Again we use the same dataset as in the theoretical course:

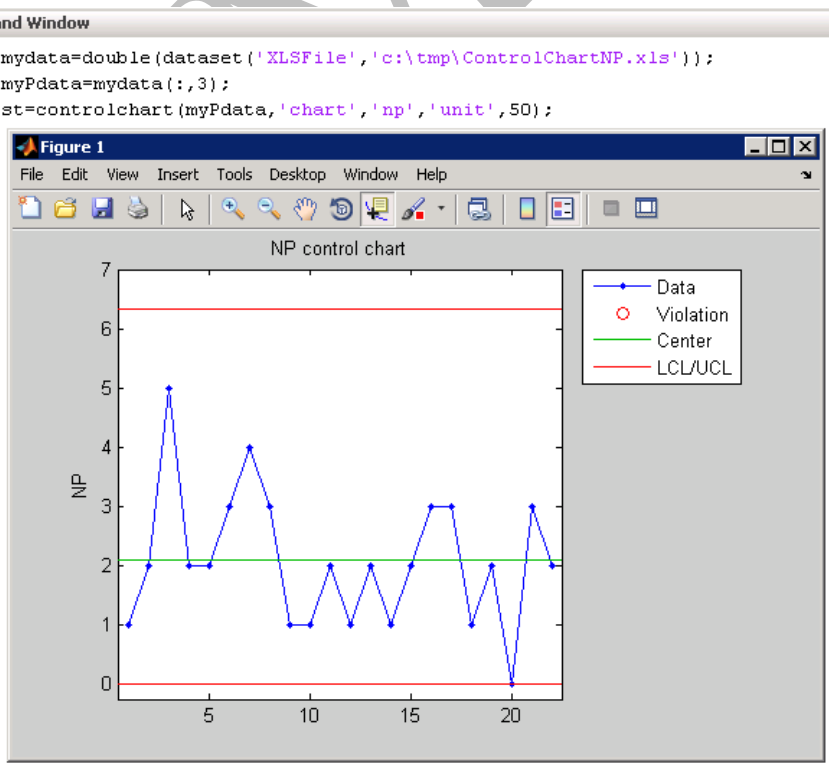
```
Command Window
>> mydata=dataset('XLSFile','c:\tmp\ControlChartNP.xls')

mydata =

    N_Lot    TailleLot    NonConformes
     1        50           1
     2        50           2
     3        50           5
     4        50           2
     5        50           2
     6        50           3
     7        50           4
     8        50           3
     9        50           1
    10        50           1
    11        50           2
    12        50           1
    13        50           2
    14        50           1
    15        50           2
    16        50           3
    17        50           3
    18        50           1
    19        50           2
    20        50           0
    21        50           3
    22        50           2
```

We get:

```
Command Window
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartNP.xls'));
>> myPdata=mydata(:,3);
>> st=controlchart(myPdata,'chart','np','unit',50);
>>
```



Nice but the lower limit can be interpreted as false...

C attribute CC

Again we use the same dataset as in the theoretical course:

```
Command Window
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartC.xls'))

mydata =

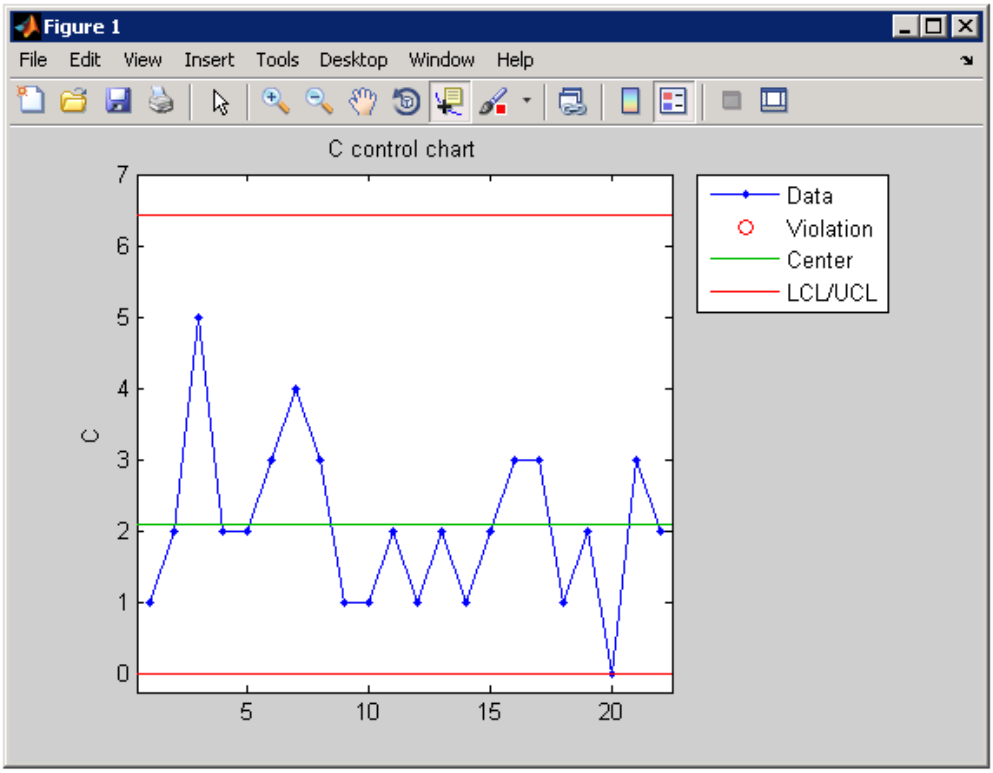
     1    50     1
     2    50     2
     3    50     5
     4    50     2
     5    50     2
     6    50     3
     7    50     4
     8    50     3
     9    50     1
    10    50     1
    11    50     2
    12    50     1
    13    50     2
    14    50     1
    15    50     2
    16    50     3
    17    50     3
    18    50     1
    19    50     2
    20    50     0
    21    50     3
    22    50     2

fx >>
```

We get:

SAMPLE

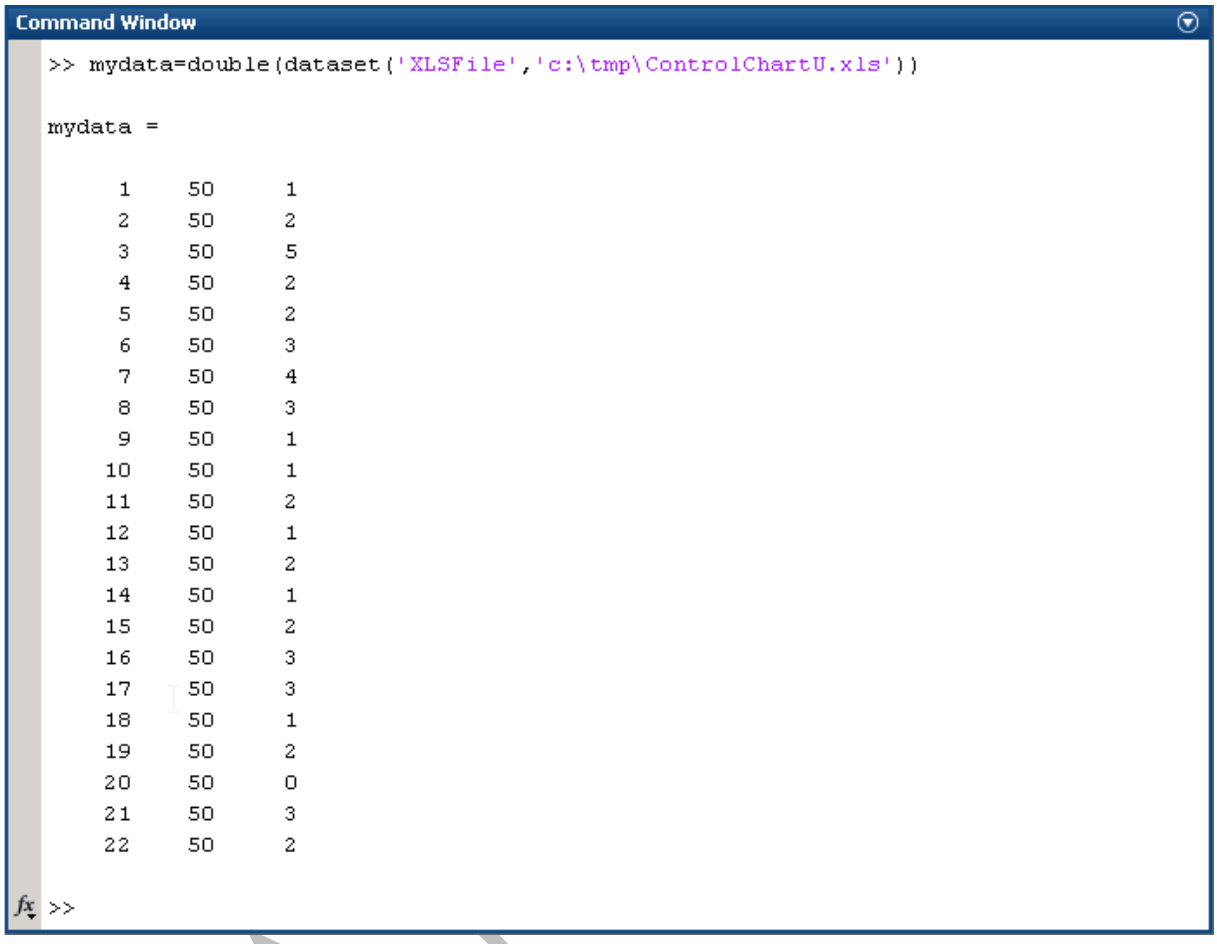
```
Command Window
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartC.xls'));
>> st=controlchart(myPdata,'chart','c','unit',50);
>>
```



SAMPLE

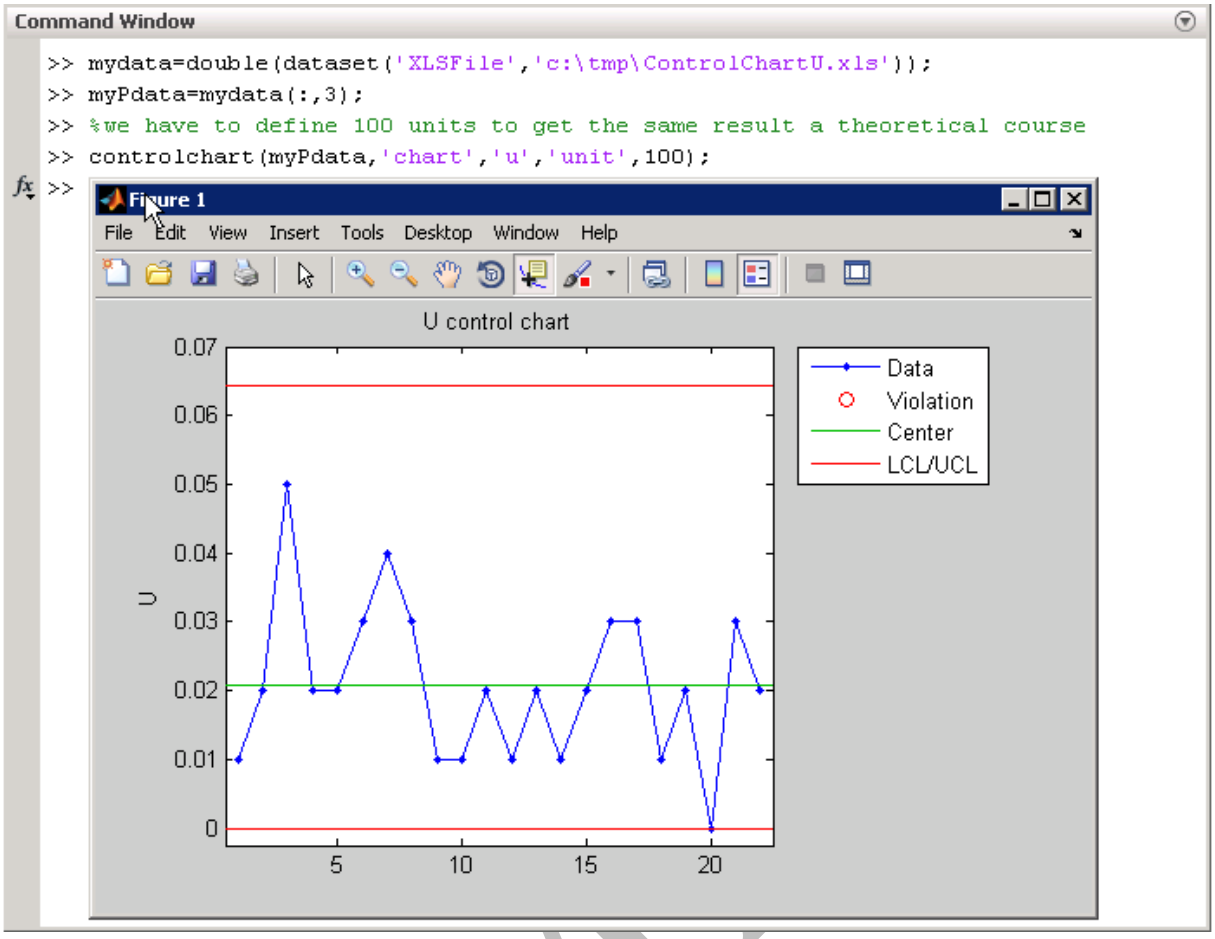
U attribute CC

Again we use the same dataset as in the theoretical course:



We get:

SAM

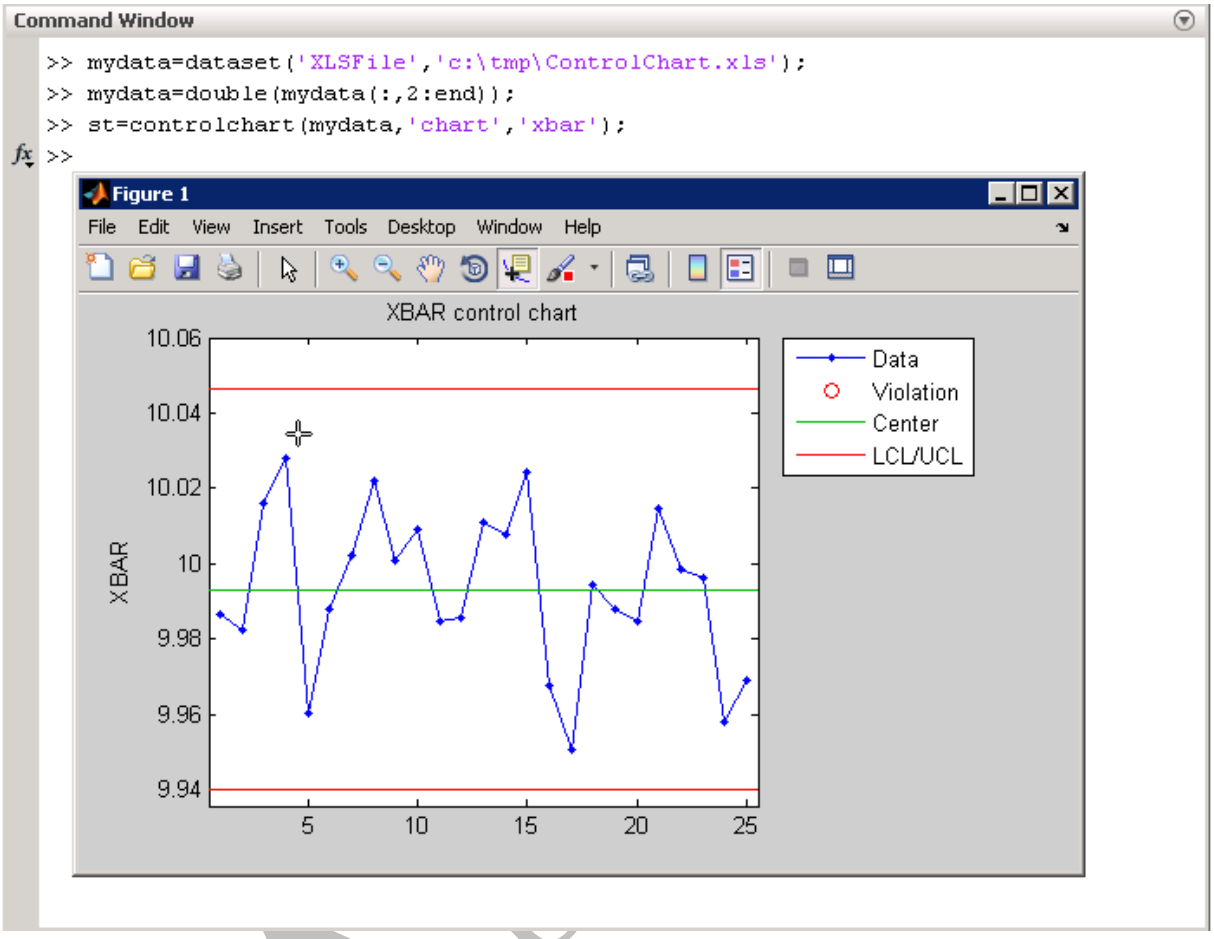


Nice but the lower limit can also be interpreted as false...

SAMPLE

Xbar-R continuous CC

We begin with the most famous control chart the Xbar-R. Once we have imported the data we use the function **controlchart( )** and we get:

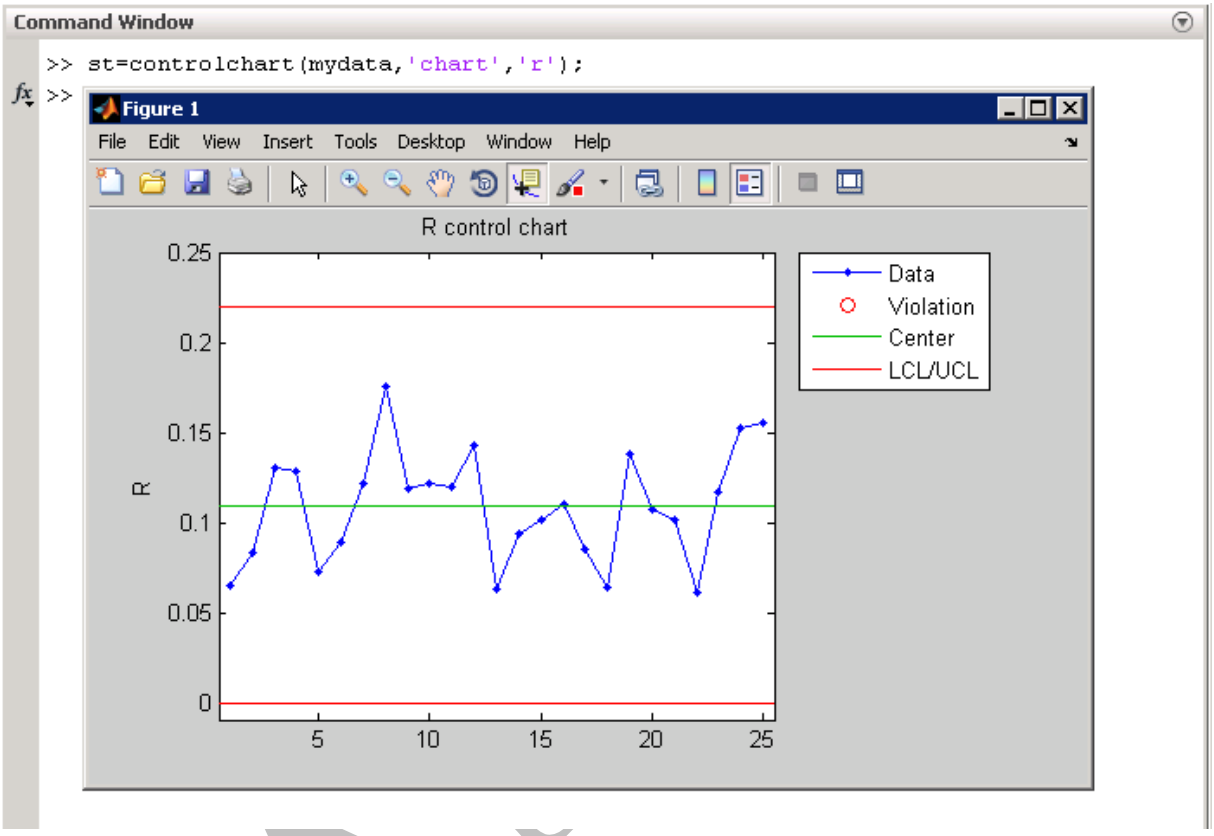


and we can see that we get exactly the same result as the one calculated by hand in the theoretical Industrial Engineering course.

Rbar-R continuous CC

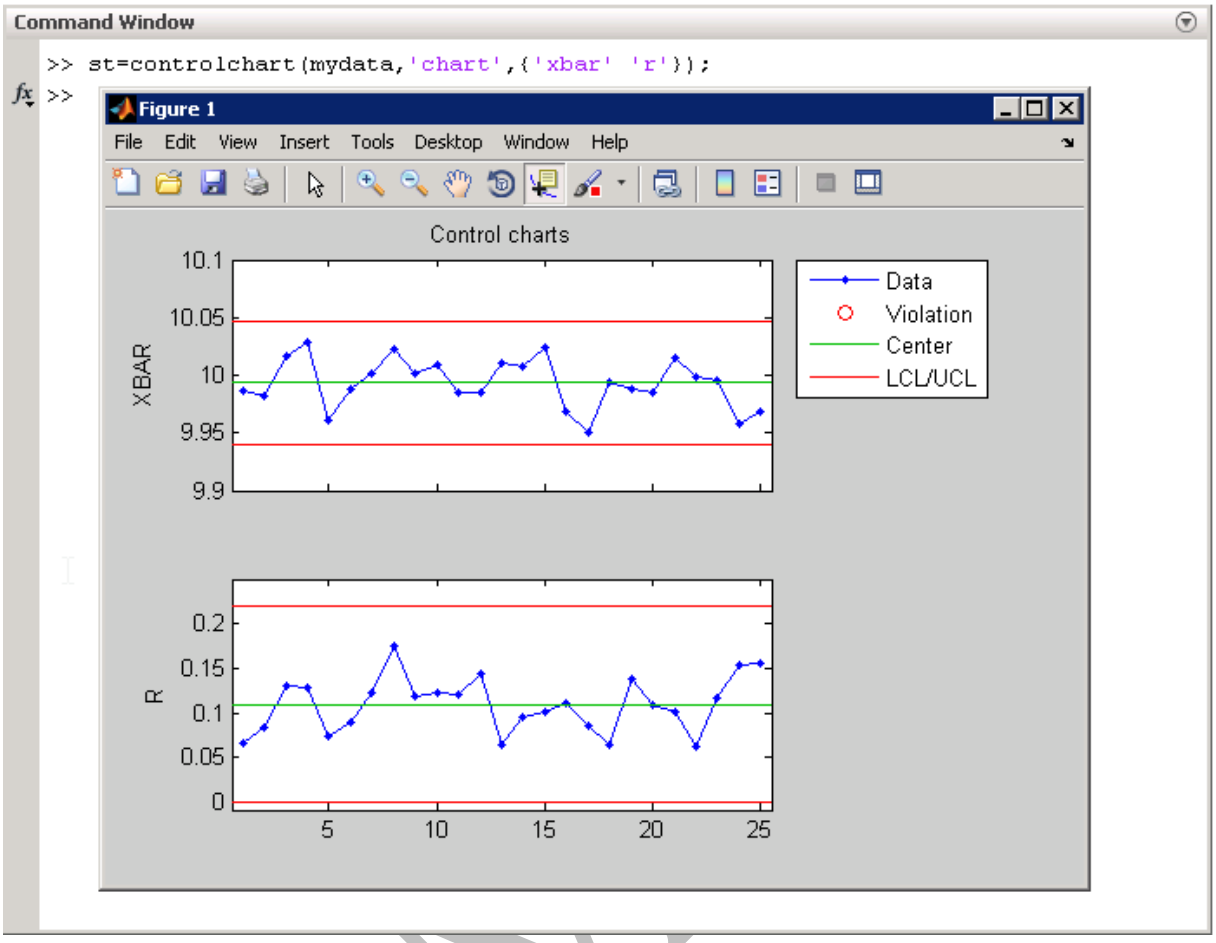
A less common bar chart than the previous one is the the Rbar-R control chart. Here we will also use the same dataset as before to see if we obtain the same result as in the theoretical course.

We always use the **controlchart( )** function:



We get exactly the same values as those calculated by hand.

If we want the both chart on the same figure we use the same function again:

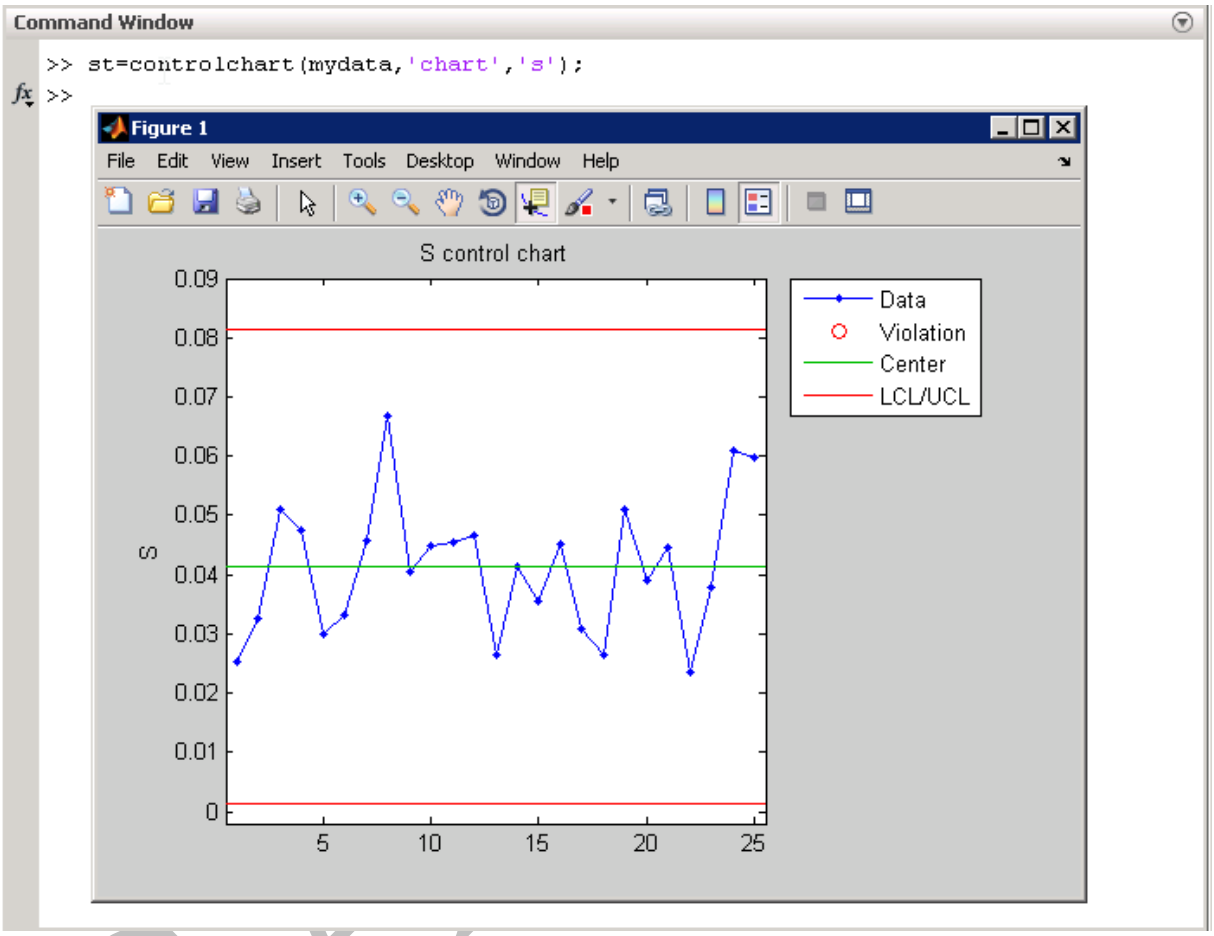


SAM

Sbar-S continuous CC

Also a less common bar chart than the the Xbar and Rbar one is the the Sbar-S control chart. Here we will also use the same dataset as before to see if we obtain the same result as in the theoretical course.

We always use the **controlchart( )** function:



We get exactly the same values as those calculated by hand.

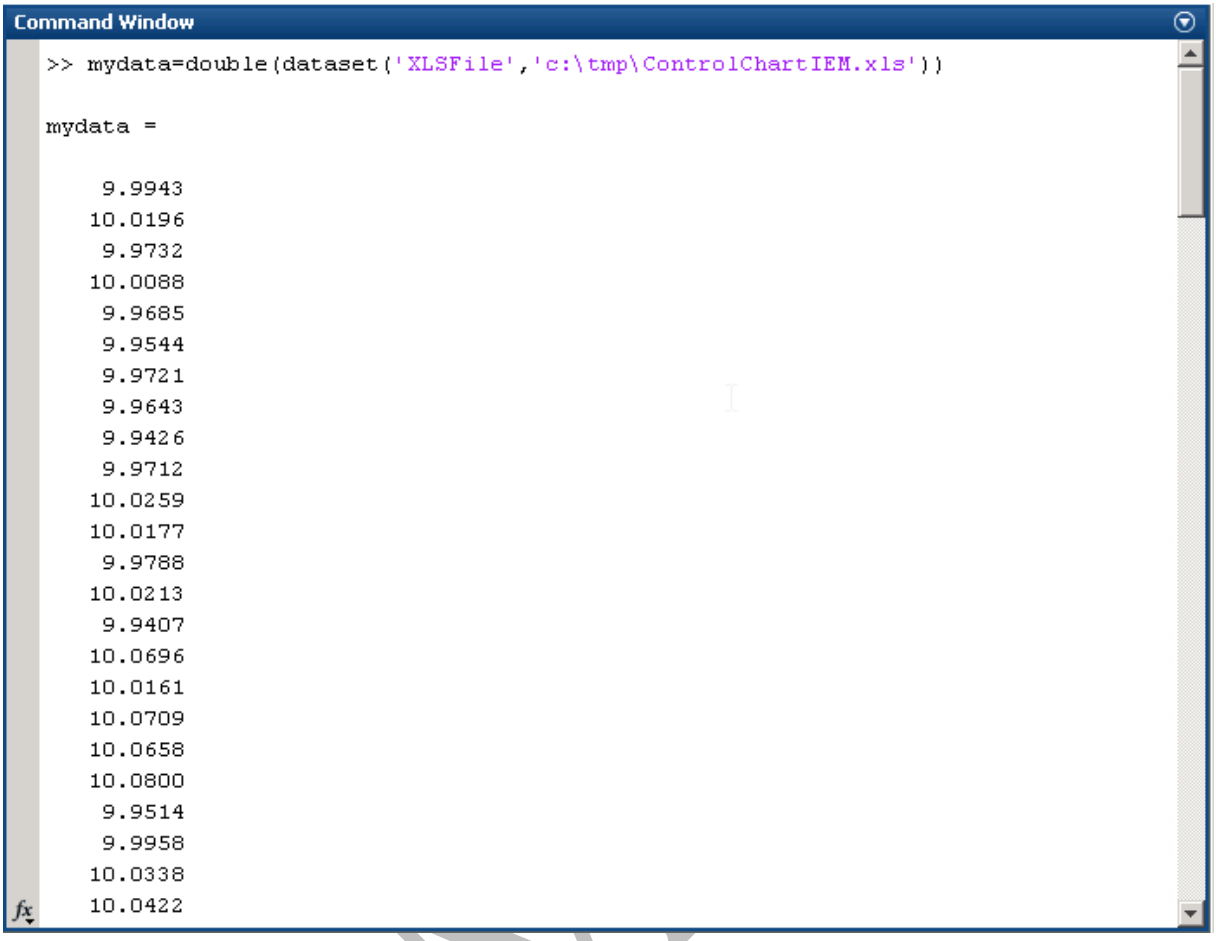
I-EM continuous CC

Individual charts are use a lot in small production process. We will check here if we also get the same individual moving average range control chart as in the theoretical course:

↓	9.9943	10.0196	9.9732	10.0088	9.9685	9.9544
	9.9721	9.9643	9.9426	9.9712	10.0259	10.0177
	9.9788	10.0213	9.9407	10.0696	10.0161	10.0709
	10.0658	10.08	9.9514	9.9958	10.0338	10.0422
	10.0136	9.9791	9.9449	9.9403	9.9424	9.9421
	9.9495	10.0057	9.9715	10.0388	10.0019	9.9616
	9.9144	10.0321	10.0361	10.0253	10.0109	9.9935
	9.9748	9.9962	10.0369	10.15	9.9996	9.9741
	10.0063	10.0749	9.9971	9.978	9.9934	9.9555
	10.084	9.9619	10.0063	10.0364	9.9907	9.9762
	9.9436	10.0638	9.9617	10.0138	9.9672	9.9591
	10.0074	9.9817	9.9753	10.0541	9.9845	9.9107
	10.0069	10.0282	10.0385	9.9747	10.0323	9.986
	9.9836	10.063	9.9824	9.9909	9.9684	10.0575
	10.0317	10.0485	9.9659	10.0675	10.0047	10.0277
	9.9756	9.9174	9.9076	9.9909	10.018	9.9979
	9.9818	9.8962	9.9614	9.9612	9.9694	9.9349
	10.0168	10.0043	10.0029	9.9549	10.0199	9.9691
	9.9857	10.0503	10.0381	9.9811	9.9603	9.9123
	9.9878	9.9841	9.9496	9.9754	9.9532	10.0576
	10.0657	9.9838	9.9636	9.9779	10.0423	10.0547
	9.9564	10.0184	9.989	9.9999	10.0146	10.0145
	10.0611	9.9443	9.9899	9.9833	9.9947	10.0033
	9.9054	9.9049	9.987	10.058	9.9156	9.9782
	9.9768	9.9954	9.9196	9.9341	9.9162	10.0721

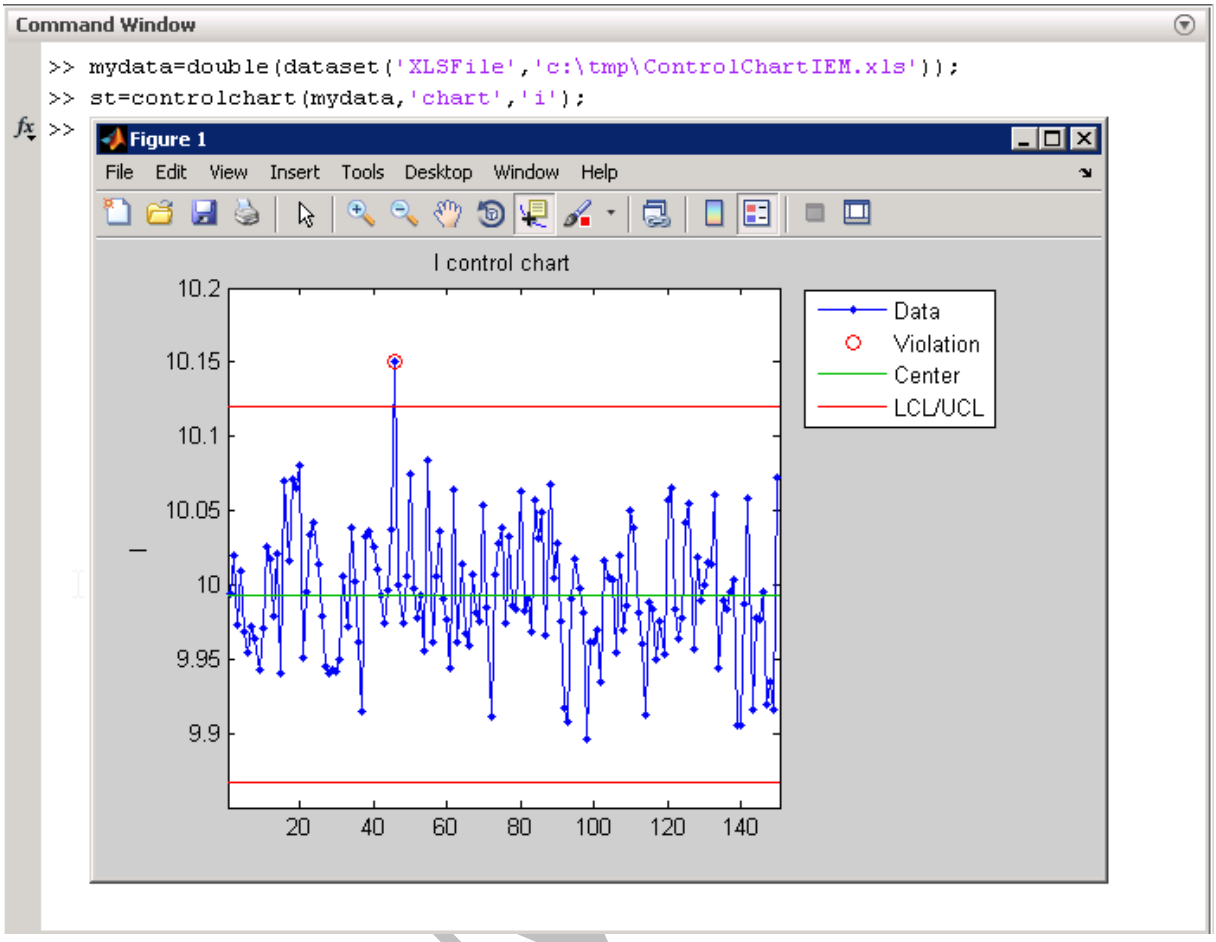
SAMPLE

We first import the data as usual:



SAM

Once the data are imported we use once again the function **controlchart( )** but with a special parameter:



We get exactly the same values as those calculated by hand.

SAM

Moving Average continuous CC

We saw in the theoretical course that Moving Average control charts belongs to the family of auto correlated (time series like) control charts.

We will also here use the same data as in the theoretical course:

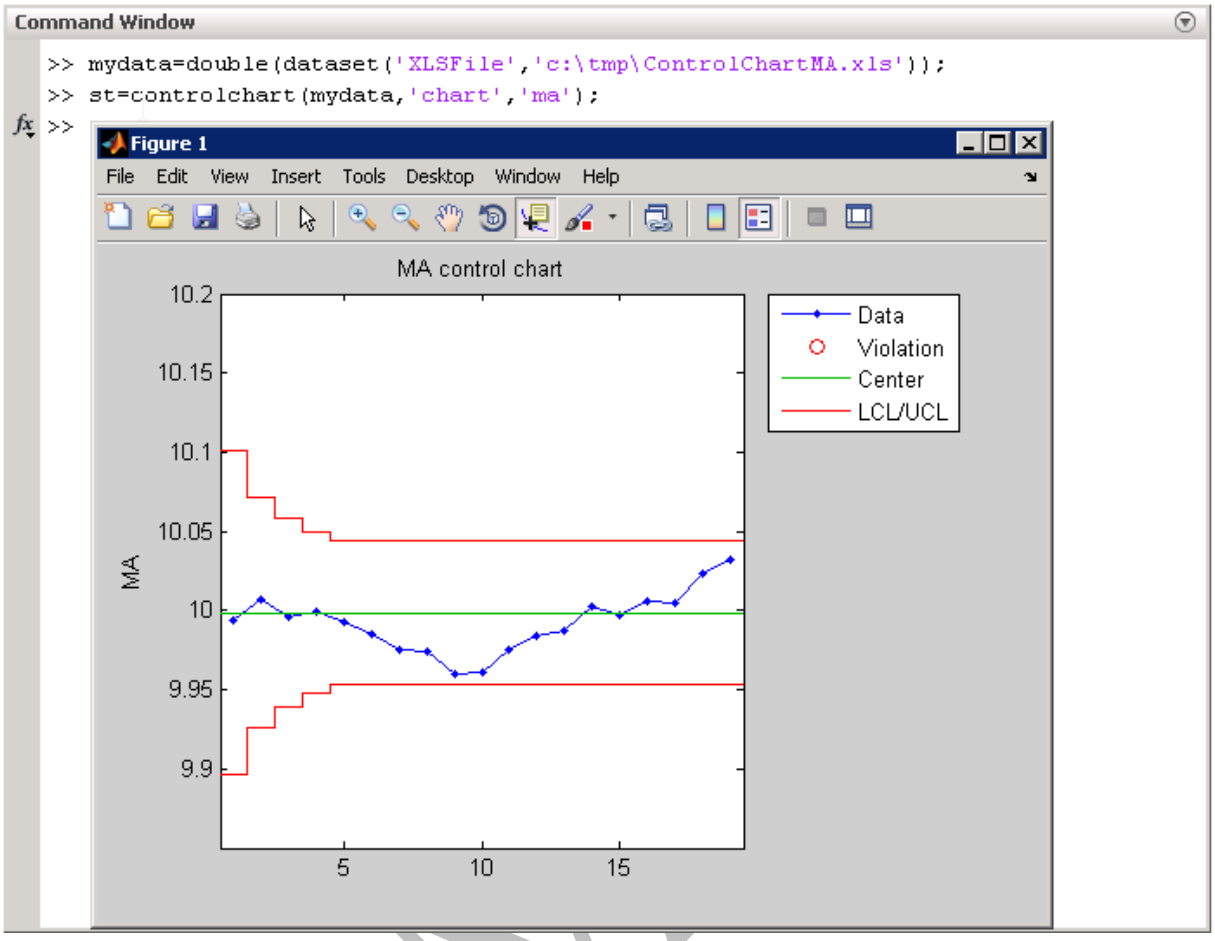
```
Command Window
>> mydata=dataset('XLSFile','c:\tmp\ControlChartMA.xls')

mydata =

    Pieces
    9.9943
     10.02
    9.9732
    10.009
    9.9685
    9.9544
    9.9721
    9.9643
    9.9426
    9.9712
    10.026
    10.018
    9.9788
    10.021
    9.9407
     10.07
    10.016
    10.071
    10.066

fx >>
```

We use again the function **controlchart( )** with a specific argument. We then get:



Here also we get the same values as in the theoretical course except for the first three point.

SAMPLE

Hotelling's bivariate ellipse control chart

As we have seen in detail in the theoretical course, this is a beautiful application of Hotelling statistics and Single Value Decomposition theorem. Actually there is no integrated function for the ellipse bivariate chart so we have to do as script (at the same time it is a good example to create an ellipse without the PDE toolbox):

```
clear all;close all;

%Create some random data
std=[rand*10 rand*10]; %generate two standard deviations
mu=[rand*10 rand*10]; %generate two means
N=300; %Number of points
%Generate to
y1=normrnd(mu(1),std(1),N,1); %Generate 1st random vector Nx1
y2=normrnd(mu(2),std(2),N,1);
data=[y1 y2];

%Calculate the eigenvectors and eigenvalues
covariance = cov(data);
[eigenvec, eigenval ] = eig(covariance);

%Get the index of the largest eigenvector
[largest_eigenvec_ind_c, r]=find(eigenval == max(max(eigenval)));
largest_eigenvec=eigenvec(:, largest_eigenvec_ind_c);

%Get the largest eigenvalue
largest_eigenval=max(max(eigenval));

%Get the smallest eigenvector and eigenvalue
if(largest_eigenvec_ind_c == 1)
    smallest_eigenval = max(eigenval(:,2))
    smallest_eigenvec = eigenvec(:,2);
else
    smallest_eigenval = max(eigenval(:,1));
    smallest_eigenvec = eigenvec(1,:);
```

```
    end

    % Calculate the angle between the x-axis and the largest eigenvector
    %angle = atan2(largest_eigenvec(2), largest_eigenvec(1))
    angle=0.5*atan2(2*covariance(1,2),(covariance(1,1)-covariance(2,2)))

    % The angle of atan2 is between -pi and pi
    % (this of atan between -pi/2 and pi/2)
    % Let's shift it such that the angle is between 0 and 2pi
    if(angle< 0)
        angle = angle-pi;
    end
    % Get the coordinates of the data mean
    avg = mean(data);

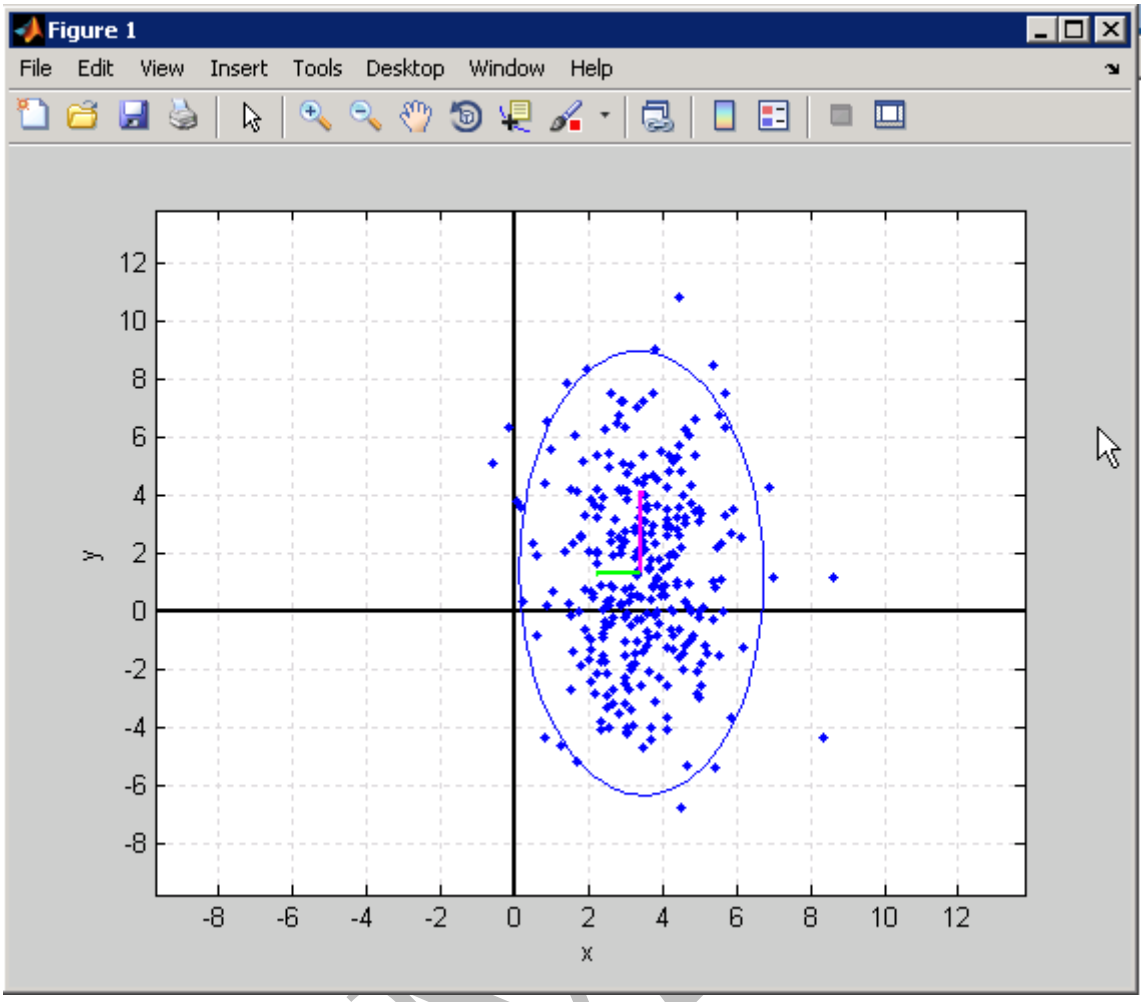
    % Get the 95% confidence interval error ellipse
    chisquare_val = 2.4477;
    theta_grid = linspace(0,2*pi);
    phi = angle;
    X0=avg(1);
    Y0=avg(2);
    a=chisquare_val*sqrt(largest_eigenval);
    b=chisquare_val*sqrt(smallest_eigenval);

    %the ellipse in x and y coordinates
    ellipse_x_r=a*cos( theta_grid );
    ellipse_y_r=b*sin( theta_grid );
```

SAMPLE

```
57
58      %define a rotation matrix
59 -    R=[ cos(phi) sin(phi); -sin(phi) cos(phi) ];
60
61      %let's rotate the ellipse to some angle phi
62 -    r_ellipse=[ellipse_x_r;ellipse_y_r]' * R;
63
64      %draw the error ellipse
65 -    plot(r_ellipse(:,1) + X0,r_ellipse(:,2) + Y0,'-')
66 -    hold on;
67
68      %dlot the original data
69 -    plot(data(:,1), data(:,2), '.');
70 -    mindata = min(min(data));
71 -    maxdata = max(max(data));
72 -    xlim([mindata-3, maxdata+3]);
73 -    ylim([mindata-3, maxdata+3]);
74 -    hold on;
75
76      % Plot the eigenvectors
77 -    quiver(X0, Y0, largest_eigenvec(1)*sqrt(largest_eigenval), ...
78          largest_eigenvec(2)*sqrt(largest_eigenval), '-m', 'LineWidth',2);
79 -    quiver(X0, Y0, smallest_eigenvec(1)*sqrt(smallest_eigenval), ...
80          smallest_eigenvec(2)*sqrt(smallest_eigenval), '-g', 'LineWidth',2);
81 -    hold on;
82
83      % Set the axis labels
84 -    hXLabel = xlabel('x');
85 -    hYLabel = ylabel('y');
87 -    grid on;
88 -    set(gca,'Xcolor',[0.85 0.85 0.85]);
89 -    set(gca,'Ycolor',[0.85 0.85 0.85]);
90 -    c_axes=copyobj(gca,gcf);
91 -    set(c_axes,'color','none','xcolor','k','xgrid','off','ycolor',...
92          'k','ygrid','off');
93 -    hx=graph2d.constantline(0,'color','k','LineWidth',2);
94 -    changedependvar(hx,'x');
95 -    hy=graph2d.constantline(0,'color','k','LineWidth',2);
96 -    changedependvar(hy,'y');
```

The result will be:


Figure 1
File Edit View Insert Tools Desktop Window Help
12
10
8
6
4
2
0
-2
-4
-6
-8
y
-8 -6 -4 -2 0 2 4 6 8 10 12
x

SAM

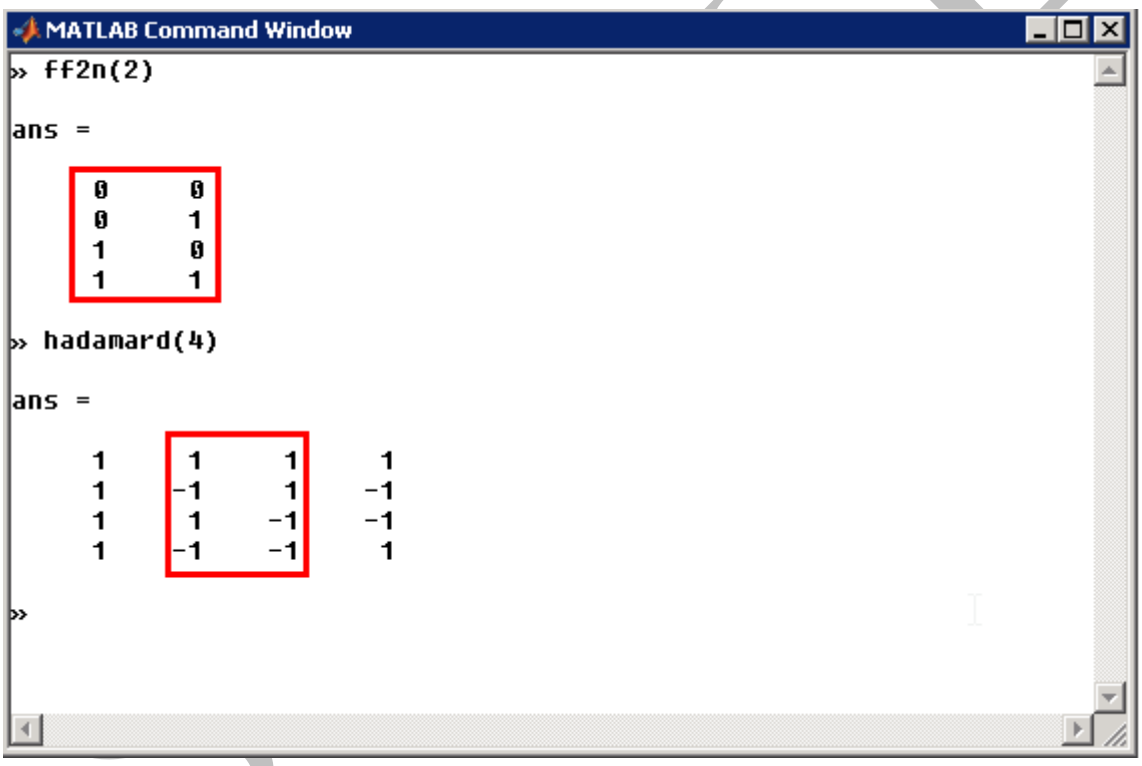
Design of experiments

As we know, designs of experiments are a huuuuge subject! MATLAB is very poor in this area to help to simplify the life of the end user. I strongly recommend to use instead softwares like JMP, Minitab, DesignExpert or R.

In the point of view of the MathWorks dev team (point of view that is fact not totally false), be able to generate a factorial design is enough for processing after the response data with all existing functions of the Statistical Toolbox. Even if it's true it's a pain in the ass for the end user this is why at least 99% of people prefer to use specialized softwares.

Two-level full factorial design

We will check if we get the some simple factorial design that we have calculated by hand in the theoretical course using the function **ff2n( )**:

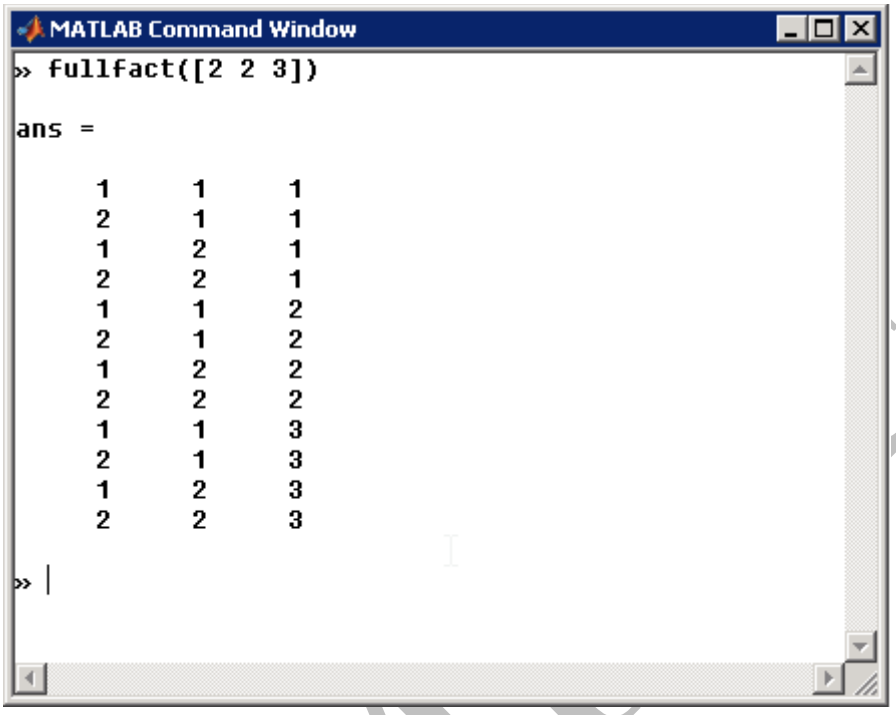


We have also use the function **hadamard( )** as you can see above to compare it with the **ff2n(2)** and see that we can conclude to the same thing as in the theoretical course.

Everything is fine here!

Full factorial design

Now have a look to the full factorial design with 3 factors of respectively 2, 2 and 3 levels. Using the function **fullfact( )** we get as expected the 12 rows.

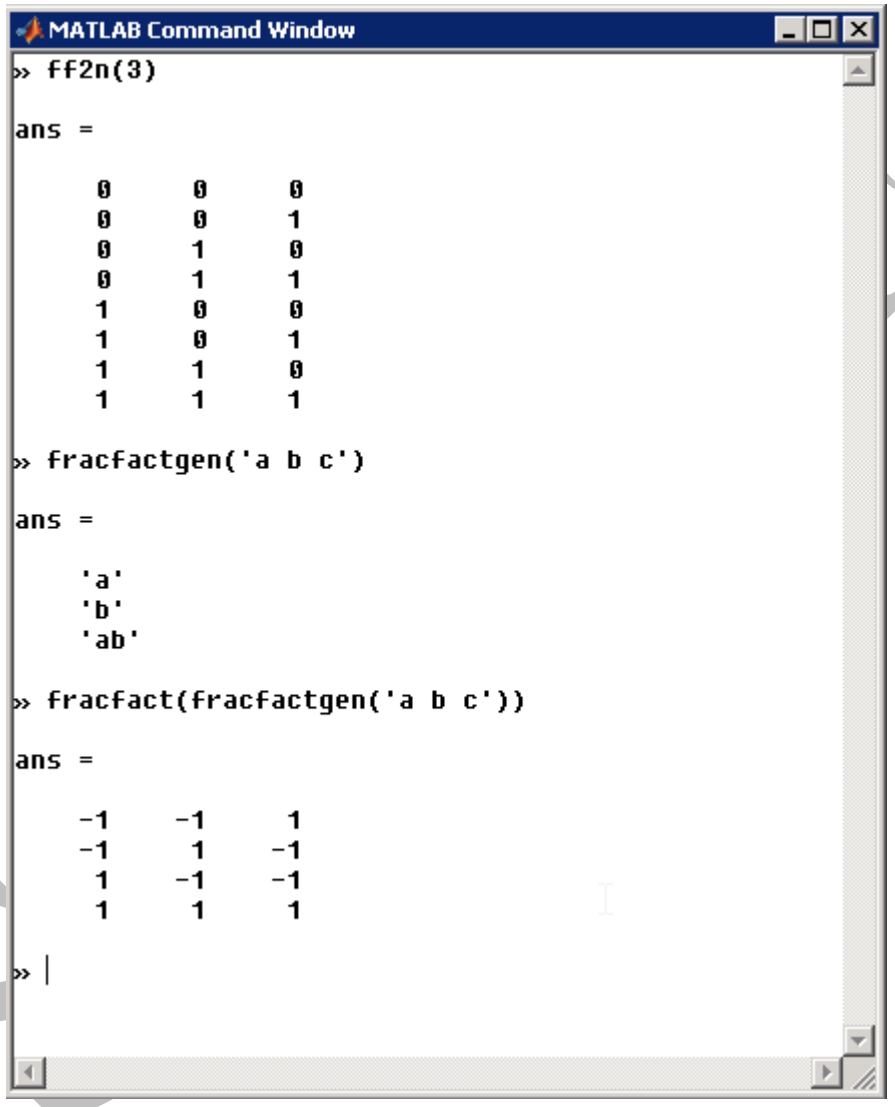


SAMPLE

Fractional factorial design

First we use the **ff2n( )** function to get the full factorial one as in the theoretical course. Once this is done, we chose the generators as we saw it in the theoretical course using the function **fracfactgen( )**.

And after only we use the **fracfact( )** function to get the fractional design we calculated by hand in the theoretical course!



Simulations techniques

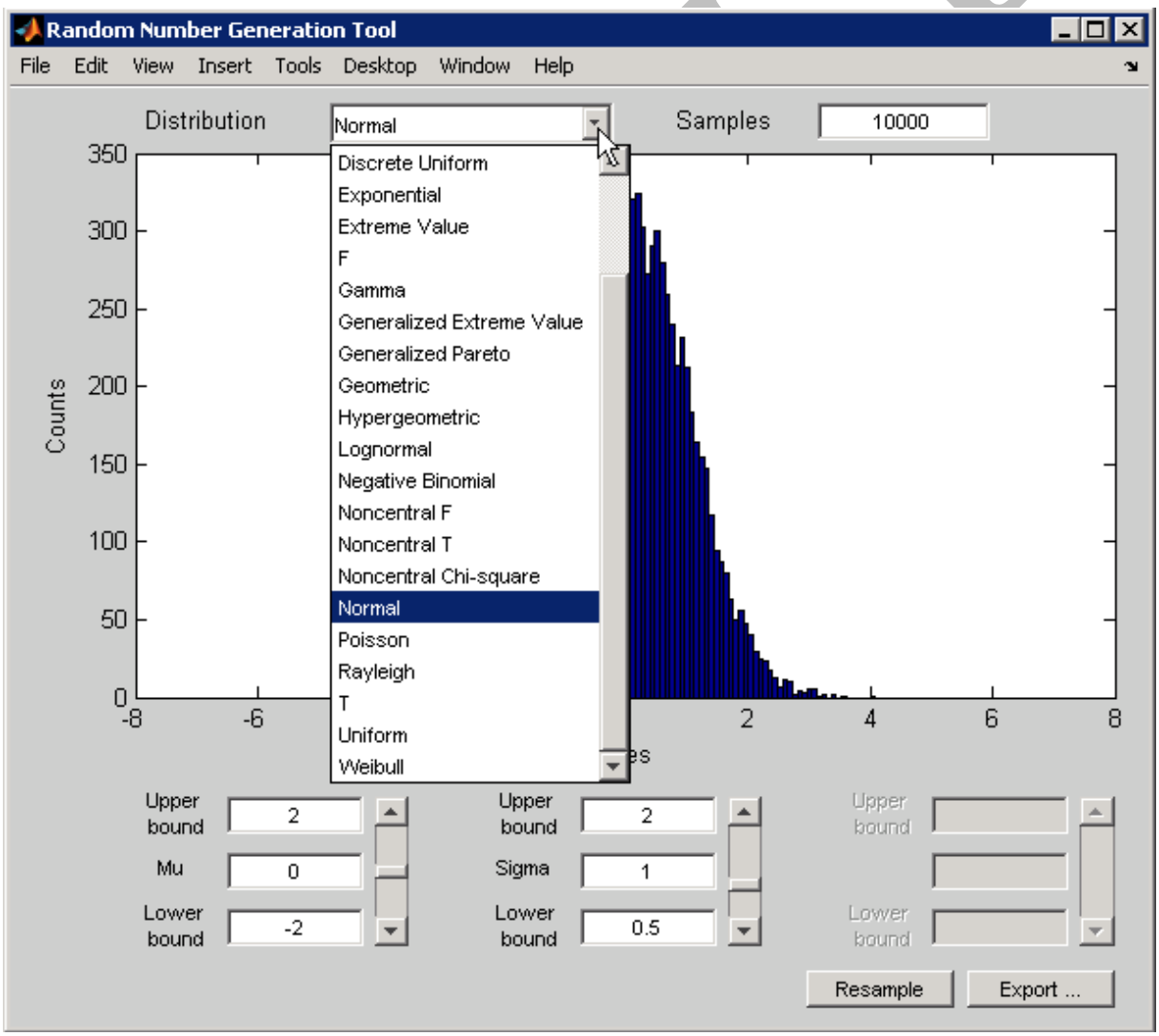
Once again we will focus in this section only about stuff that we have study in details during the theoretical course.

Randtool wizard

This tool is more for academic purpose than anything else for demonstrating to students how behaves shapes of distributions depending on the number of samples. First open the tool:



So appears:

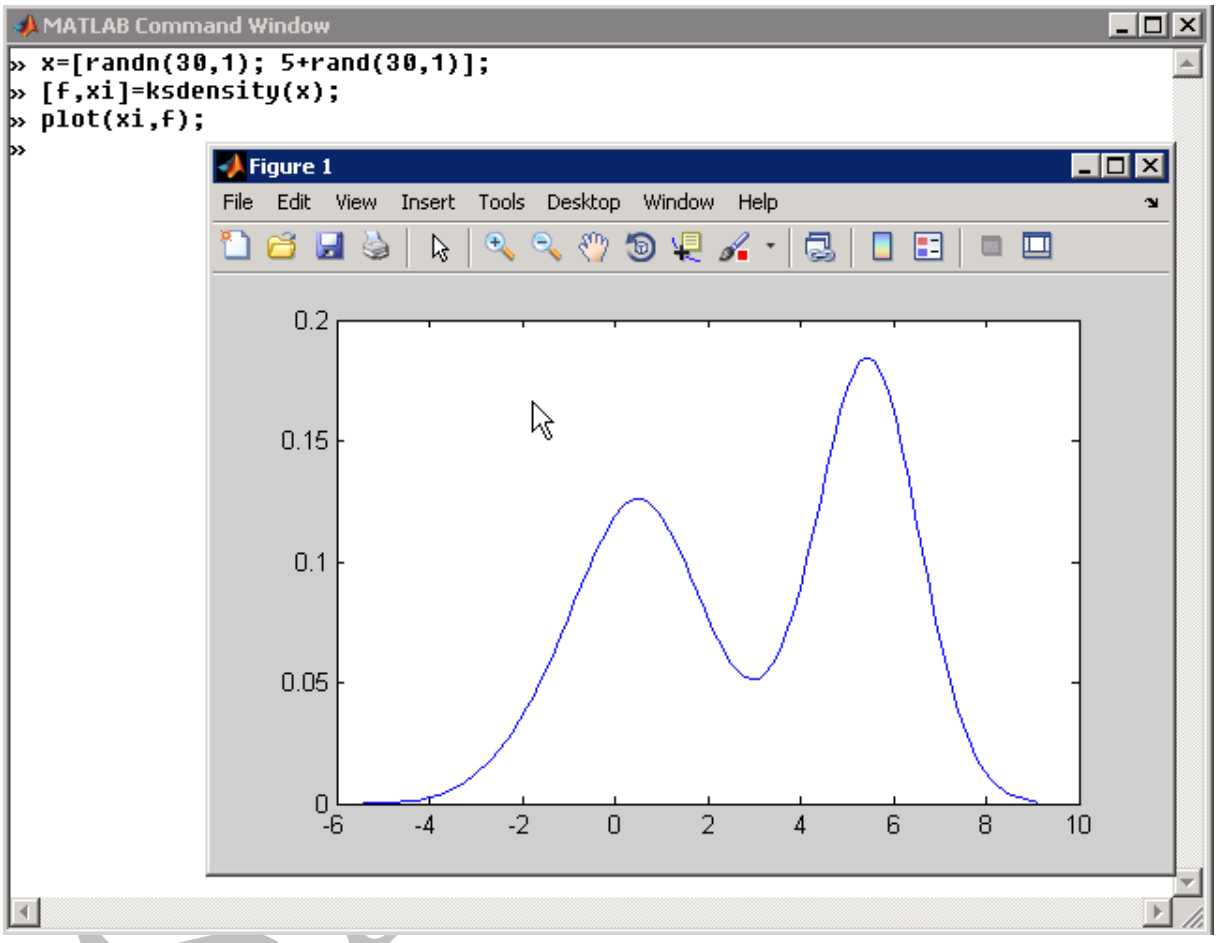


The **Upper bound** and **Lower bounds** fields visible are here to define each scrollbar limits if you want to play to change the distribution parameters with the scrollbars!

Kernel Smoothing

As we know, using a histogram to judge the shape of a distribution is a bias in itself. Using Kernel Smoothing or even better CDF is more accurate.

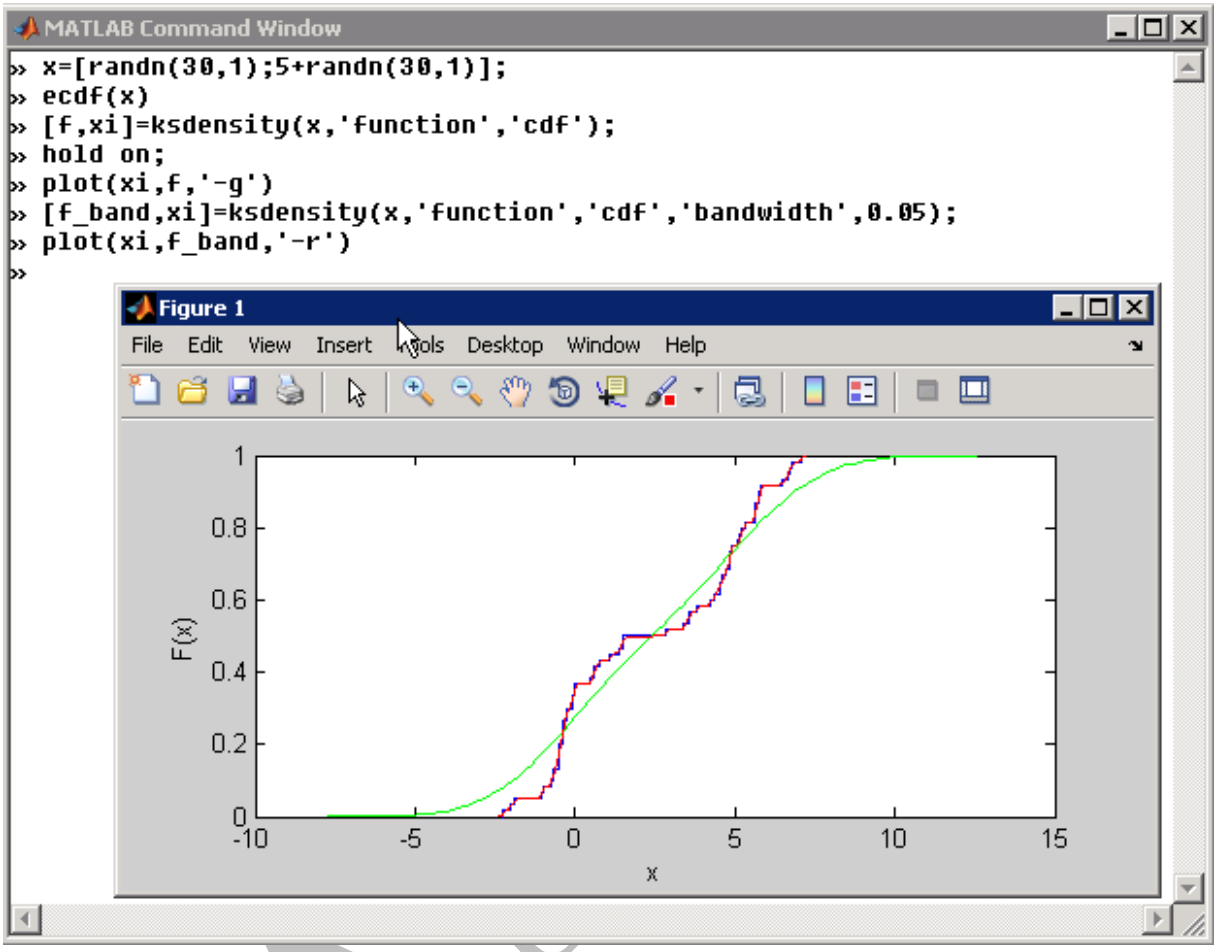
Because the purpose here is to check that Kernel Smoothing exists in MATLAB let us generate a vector of two different Normal random variable and apply the **ksdensity( )** function on it with default bandwidth and the default hypothesized distribution (aka the Normal one...):



So fine... seems to do something put in practice I'm not a fan of charts. I want numbers, numbers and numbers!

So can we retrieve any information about this previous kernel smoothing density with MATLAB? Yes we can!!

For example we compare here below the ECDF



SAMPLE

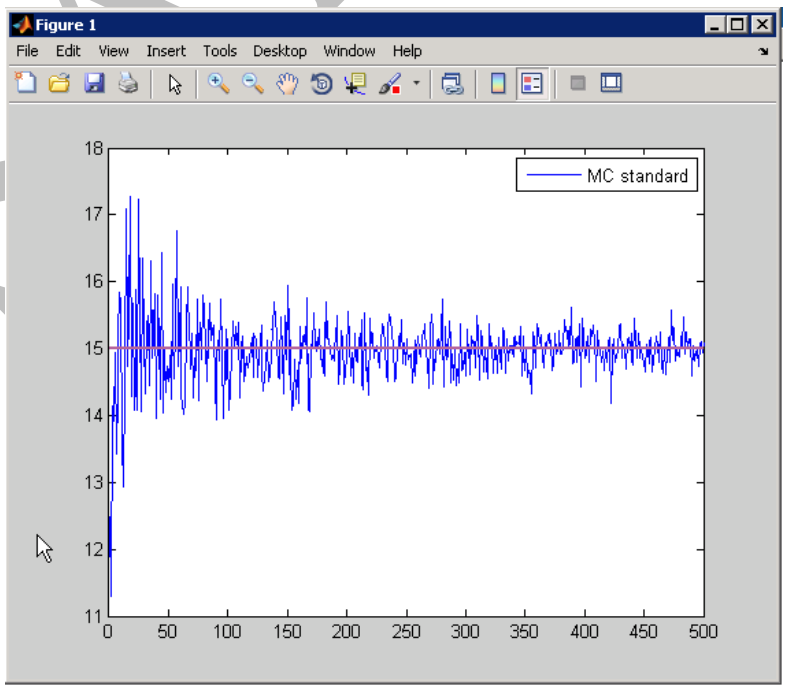
Monte Carlo standard technic

The most simple and academic example of Monte Carlo technics is to play with different random variables and calculates some of their moments.

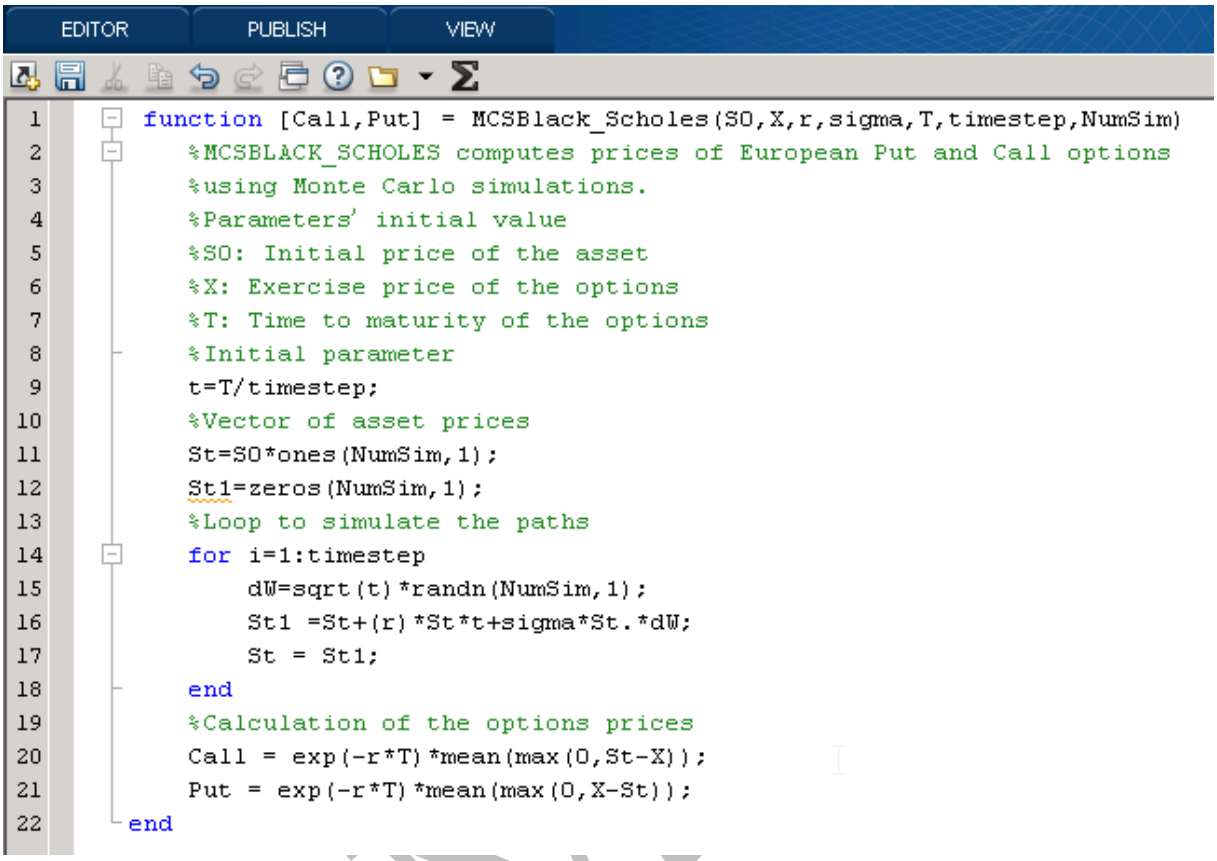
The most trivial and common case is to compute de mean of the sum of two random Normal variables because we know the Normal distribution is stable by the sum:

```
1   %standard MC method
2   clear all;clc
3   int_Simulations=500;
4   t=1:1:int_Simulations;
5   for k=1:1:int_Simulations
6       x=rand(k,1);
7       y=rand(k,1);
8       mux=5;xsigma=3;
9       muy=10;ysigma=4;
10      Nx=norminv(x,mux,xsigma); %N(5,3)
11      Ny=norminv(y,muy,ysigma); %N(10,4)
12      muNxy(k)=mean(Nx+Ny)
13  end
14  plot(t,muNxy)
15  hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
16  changedependvar(hy,'y');
17  legend('MC standard')
18
```

We can see qualitatively the convergence with the following chart:



The most known example in finance use the fundamental equation for option pricing that we proved during the theoretical finance course and at the same time this is a beautiful example of the power of vector scripting language:




```
function [Call,Put] = MCSBlack_Scholes(S0,X,r,sigma,T,timestep,NumSim)
    %MCSBLACK_SCHOLES computes prices of European Put and Call options
    %using Monte Carlo simulations.
    %Parameters' initial value
    %S0: Initial price of the asset
    %X: Exercise price of the options
    %T: Time to maturity of the options
    %Initial parameter
    t=T/timestep;
    %Vector of asset prices
    St=S0*ones(NumSim,1);
    St1=zeros(NumSim,1);
    %Loop to simulate the paths
    for i=1:timestep
        dW=sqrt(t)*randn(NumSim,1);
        St1 =St+(r)*St*t+sigma*St.*dW;
        St = St1;
    end
    %Calculation of the options prices
    Call = exp(-r*T)*mean(max(0,St-X));
    Put = exp(-r*T)*mean(max(0,X-St));
end
```

SAM

Monte Carlo with acceptance reject

As we discuss in the theoretical course of numerical methods this algorithm is quite simple to write. The script below is inspired by the original one of Dmitry Savransky:

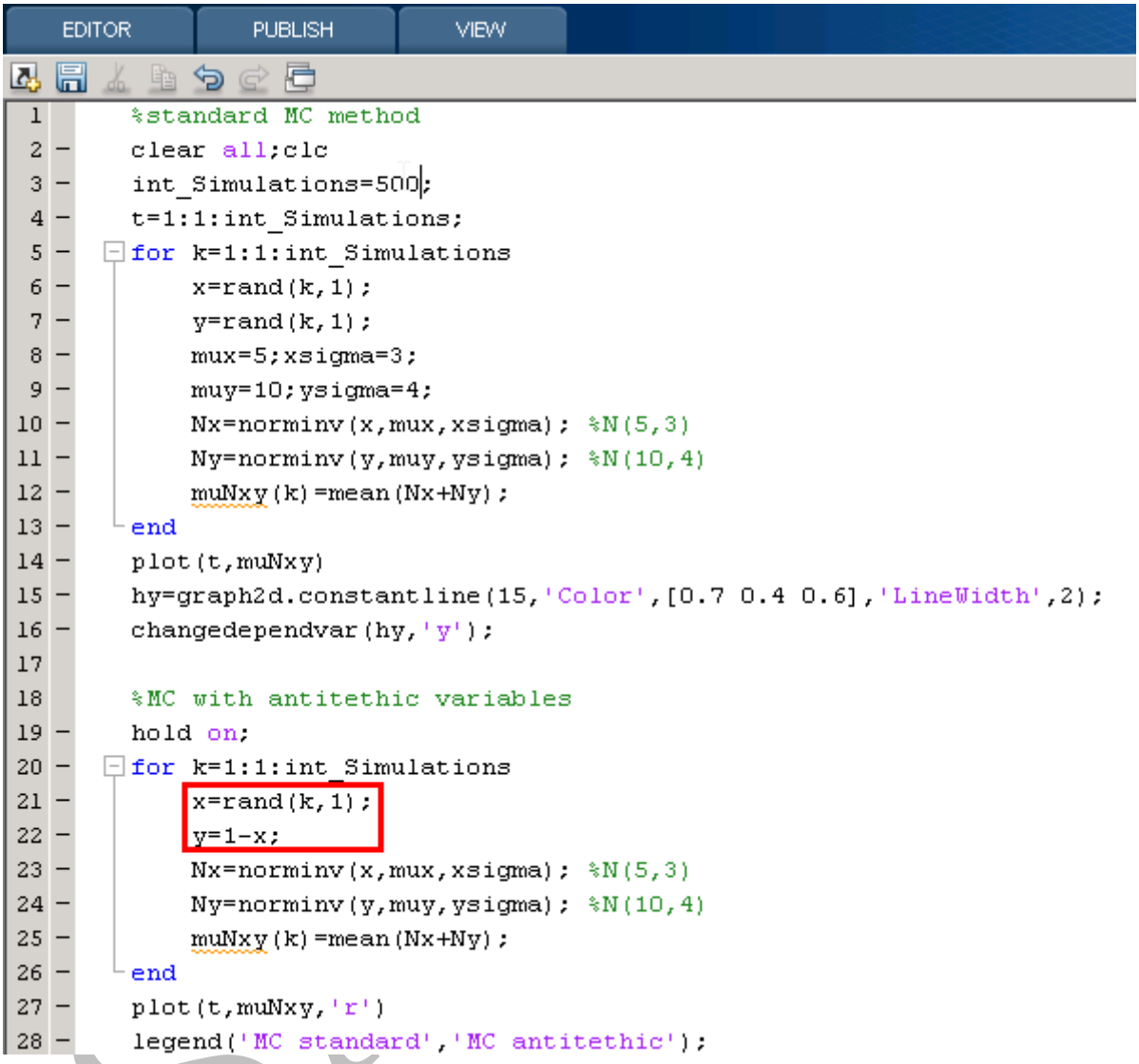
C:\sampleDist.m

EDITOR PUBLISH VIEW

```
1   function X = sampleDist(f,M,N,b,mkplt)
2   % SAMPLEDIST  Sample from an arbitrary distribution%...%
18
19  % Dmitry Savransky (dsavrans@princeton.edu)
20  % May 11, 2010
21
22  n = 0;
23  X = zeros(N,1);
24  counter = 0;
25
26  while n < N
27      %randon points on the abscisse in the interval of b
28      x = b(1) + rand(N,1)*diff(b);
29      %for each random point on the abscisse we take a point between 0 and M
30      uM = M*rand(N,1);
31      %We reject the points below our main function
32      x = x(uM < f(x)); %this result in a vector containing only non rejected values
33      if isempty(x)
34          error('No Points Sampled')
35      end
36      X(n+1:n+length(x)) = x;
37      n = n + length(x);
38  end
39      %we round to keep the N values only
40      X=X(1:N);
41
42  if exist('mkplt','var') && mkplt
42  if exist('mkplt','var') && mkplt
43      nsamps = max([min([N/1000,1000]),10]);
44      [n,c] = hist(X,nsamps);
45      bar(c,n/N*nsamps/(max(X) - min(X)))
46      hold on
47      plot(c,f(c),'r','LineWidth',2)
48      hy=graph2d.constantline(M,'Color',[0.7,0.4,0.6],'LineWidth',2);
49      changedependvar(hy,'y');
50  end
```

Monte Carlo antithetic variables technique

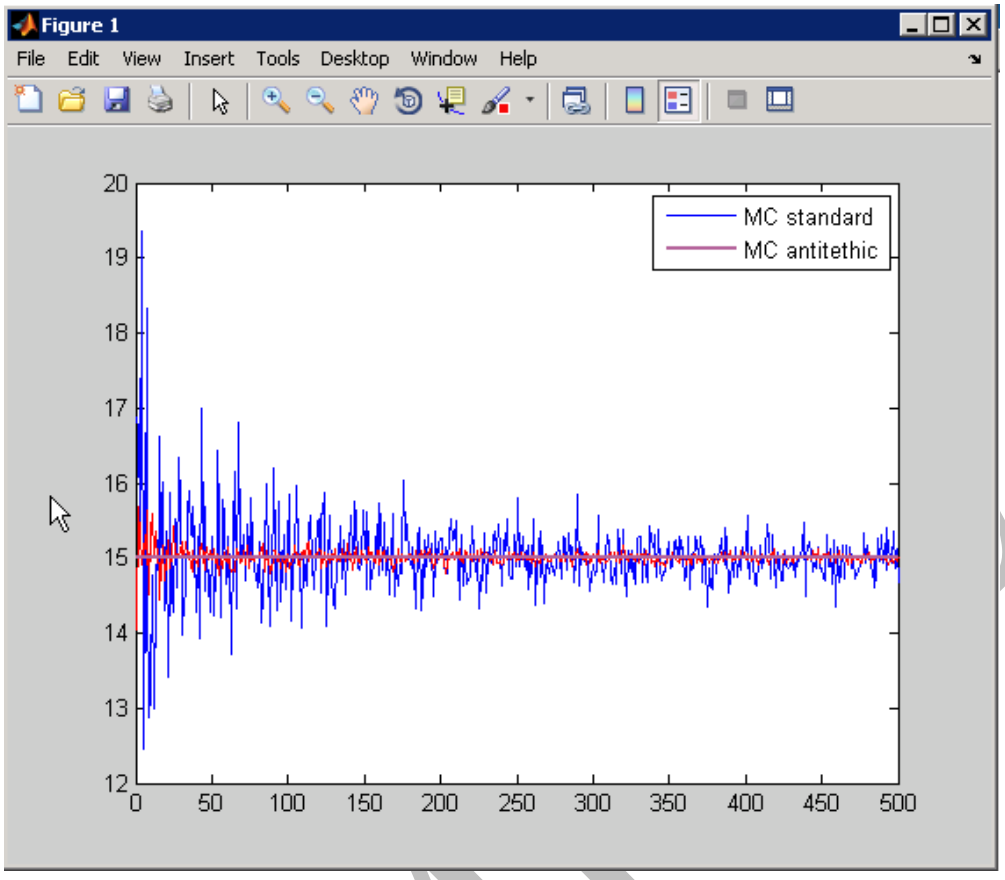
The purpose is to see the famous and simple antithetic variables technique and compare it to the standard one




```
%standard MC method
clear all;clc
int_Simulations=500;
t=1:1:int_Simulations;
for k=1:1:int_Simulations
    x=rand(k,1);
    y=rand(k,1);
    mux=5;xsigma=3;
    muy=10;ysigma=4;
    Nx=norminv(x,mux,xsigma); %N(5,3)
    Ny=norminv(y,muy,ysigma); %N(10,4)
    muNxy(k)=mean(Nx+Ny);
end
plot(t,muNxy)
hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hy,'y');

%MC with antitethic variables
hold on;
for k=1:1:int_Simulations
    x=rand(k,1);
    y=1-x;
    Nx=norminv(x,mux,xsigma); %N(5,3)
    Ny=norminv(y,muy,ysigma); %N(10,4)
    muNxy(k)=mean(Nx+Ny);
end
plot(t,muNxy,'r')
legend('MC standard','MC antitethic');
```

And as we can see on the next page, it is logic that in converge faster:



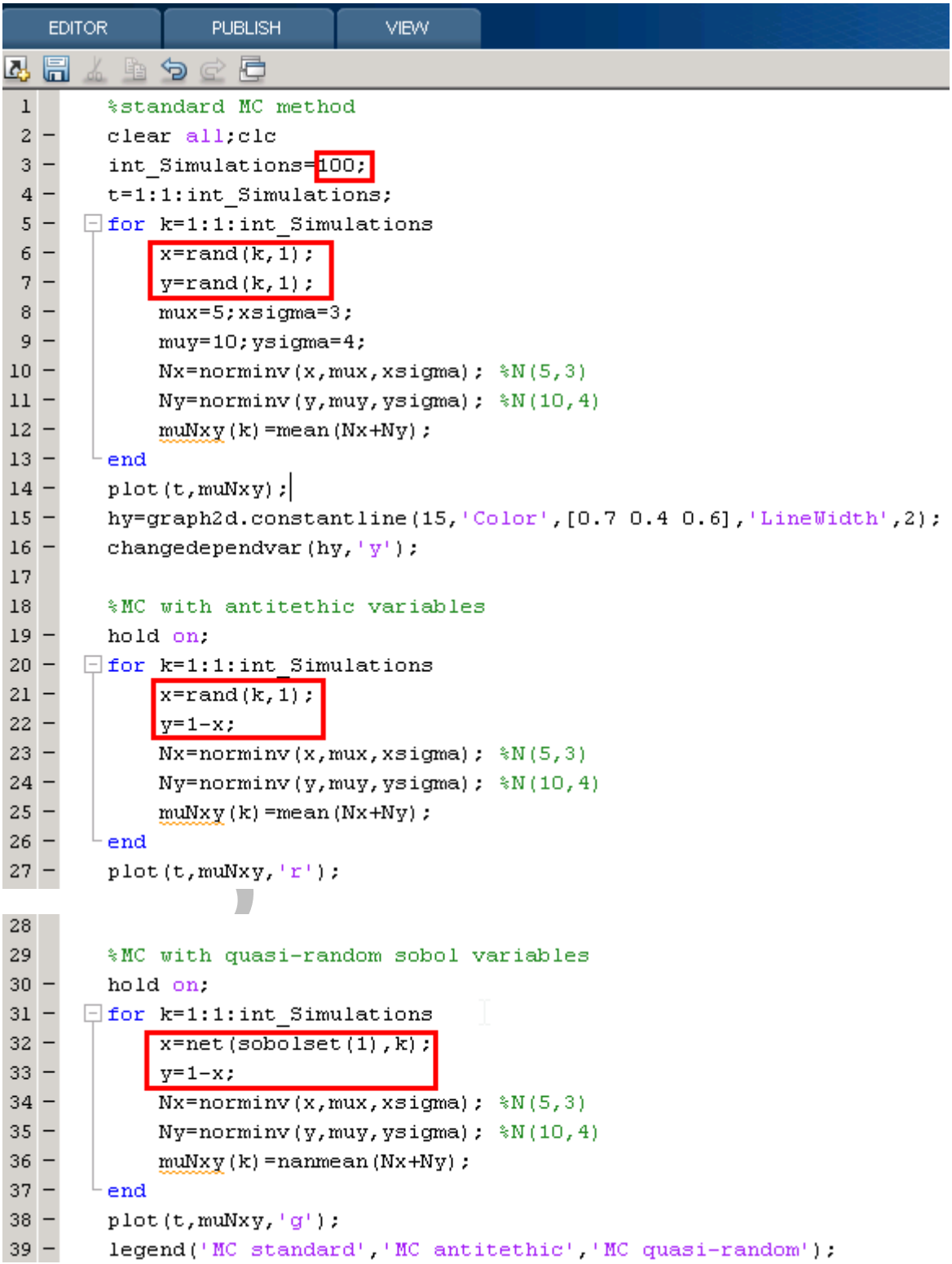
See now a famous example in finance base on the fundamental equation of option pricing. We take the same script as before but we add an antithetic variable:

EDITOR PUBLISH VIEW

```
function [Call,Put] = MCSABlack_Scholes(S0,X,r,sigma,T,timestep,NumSim)
    %MCSBLACK_SCHOLES computes prices of European Put and Call options
    %using Monte Carlo simulations.
    %Parameters' initial value
    %S0: Initial price of the asset
    %X: Exercise price of the options
    %T: Time to maturity of the options
    %Initial parameter
    t=T/timestep;
    %Vector of asset prices
    St=S0*ones(NumSim,1);
    St1=zeros(NumSim,1);
    %Loop to simulate the paths
    for i=1:timestep
        dW=sqrt(t)*randn(NumSim,1);
        St1 =St+(r)*St*t+sigma*St.*dW;
        St = St1;
    end
    %Calculation of the options prices
    Call = exp(-r*T)*mean(max(0,St-X));
    Put = exp(-r*T)*mean(max(0,X-St));
end
```

Monte Carlo with antithetic quasi-random variables technique

As we saw it in the theoretical course there a lot of quasi-random variables generators. Here we will compare the two previous method with the Sobol quasi-random generator using the function **sobolset( )**:

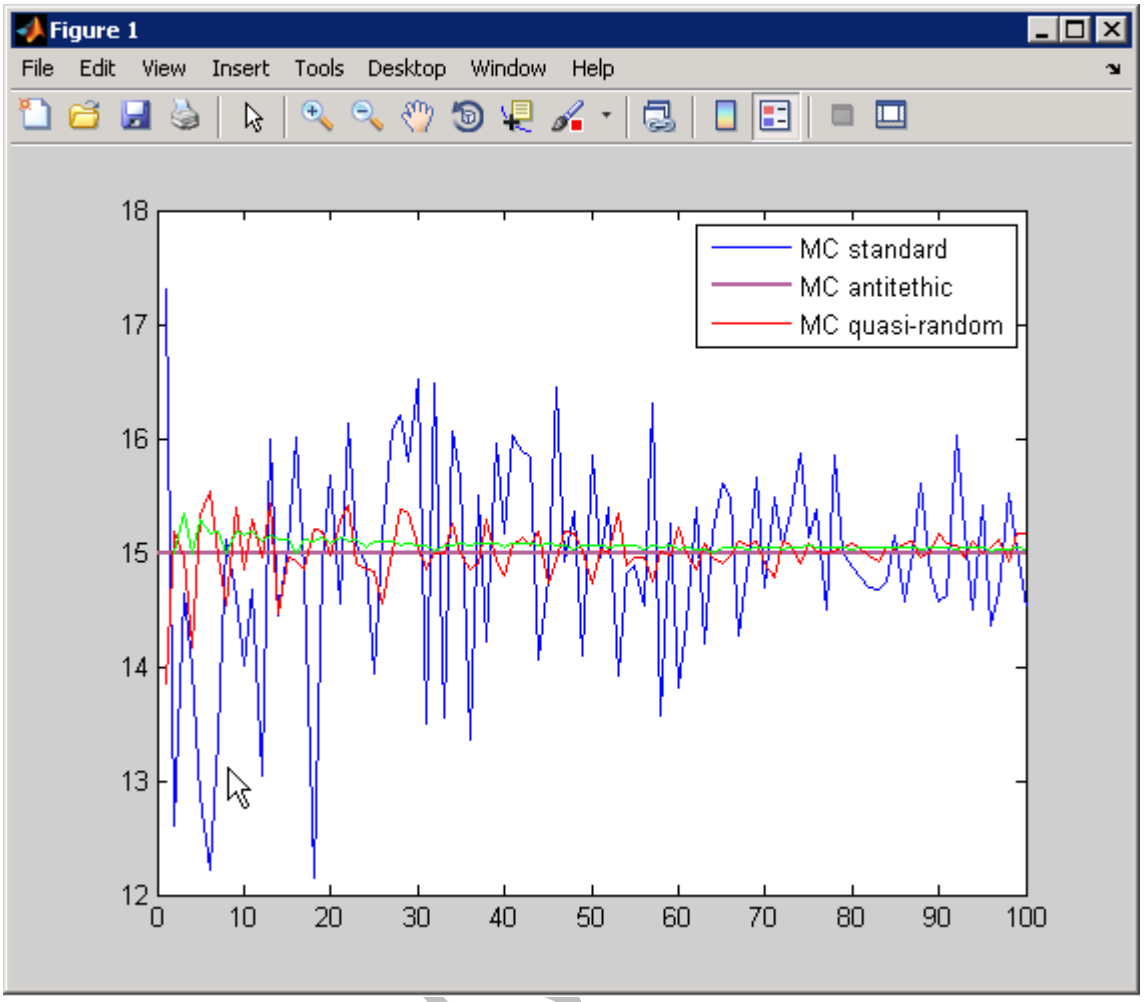



```
%standard MC method
clear all;clc
int_Simulations=100;
t=1:1:int_Simulations;
for k=1:1:int_Simulations
    x=rand(k,1);
    y=rand(k,1);
    mux=5;xsigma=3;
    muy=10;ysigma=4;
    Nx=norminv(x,mux,xsigma); %N(5,3)
    Ny=norminv(y,muy,ysigma); %N(10,4)
    muNxy(k)=mean(Nx+Ny);
end
plot(t,muNxy);
hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hy,'y');

%MC with antitethic variables
hold on;
for k=1:1:int_Simulations
    x=rand(k,1);
    y=1-x;
    Nx=norminv(x,mux,xsigma); %N(5,3)
    Ny=norminv(y,muy,ysigma); %N(10,4)
    muNxy(k)=mean(Nx+Ny);
end
plot(t,muNxy,'r');


%MC with quasi-random sobol variables
hold on;
for k=1:1:int_Simulations
    x=net(sobolset(1),k);
    y=1-x;
    Nx=norminv(x,mux,xsigma); %N(5,3)
    Ny=norminv(y,muy,ysigma); %N(10,4)
    muNxy(k)=nanmean(Nx+Ny);
end
plot(t,muNxy,'g');
legend('MC standard','MC antitethic','MC quasi-random');
```

We get:

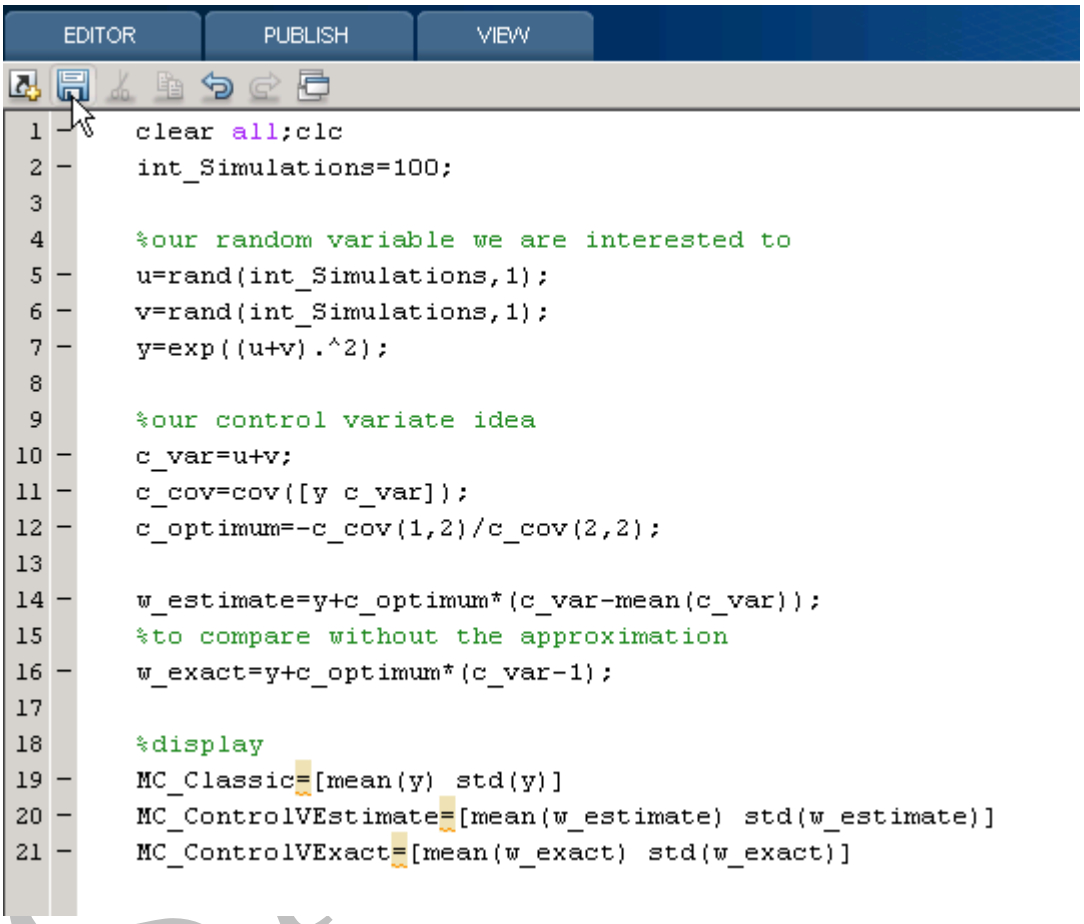

Figure 1
File Edit View Insert Tools Desktop Window Help
MC standard
MC antitethic
MC quasi-random
18
17
16
15
14
13
12
0 10 20 30 40 50 60 70 80 90 100

SAM

Monte Carlo by control variable (variance reduction technique)

We have also study during the theoretical course that there is also another reduction variance technic based on a control variable but that is in practice difficult to apply because we have to choose the right control variate expression…

The purpose here is to reproduce exactly the example that we used in the theoretical course to illustrate the method:




```
clear all;clc
int_Simulations=100;

%our random variable we are interested to
u=rand(int_Simulations,1);
v=rand(int_Simulations,1);
y=exp((u+v).^2);

%our control variate idea
c_var=u+v;
c_cov=cov([y c_var]);
c_optimum=-c_cov(1,2)/c_cov(2,2);

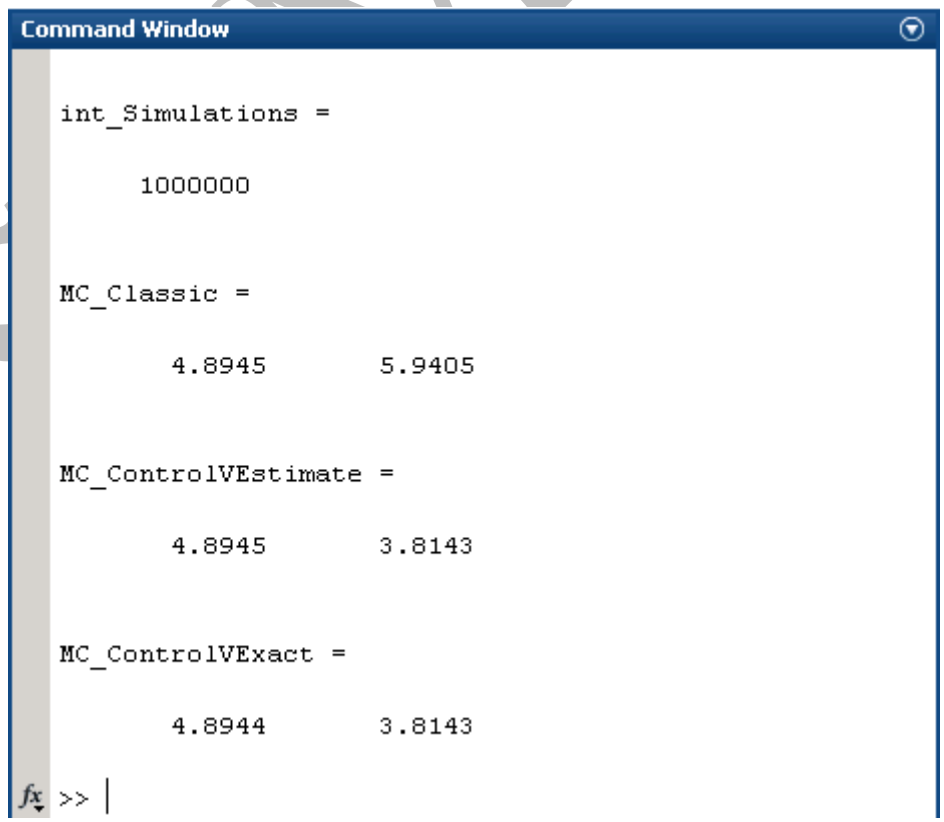
w_estimate=y+c_optimum*(c_var-mean(c_var));
%to compare without the approximation
w_exact=y+c_optimum*(c_var-1);

%display
MC_Classic=[mean(y) std(y)]
MC_ControlVEstimate=[mean(w_estimate) std(w_estimate)]
MC_ControlVExact=[mean(w_exact) std(w_exact)]
```

Once we run it with 100 samples we get:



And with 1'000'000 simulations:



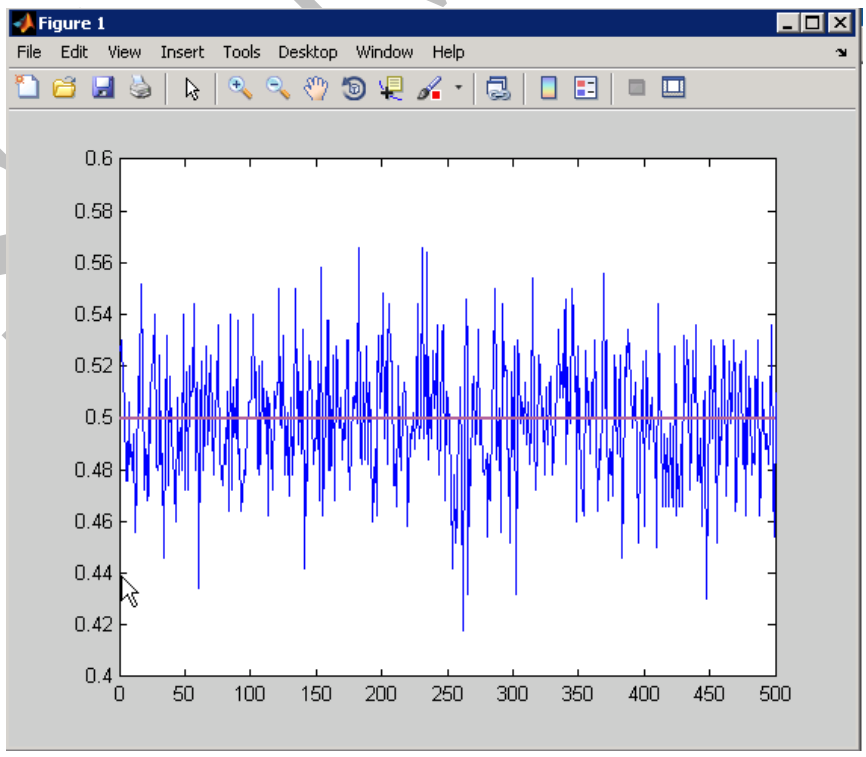
Monte Carlo importance sampling

The importance sampling in a very powerful empirical method to get typically very small values for distributions tails.

First we make a simple simulation to get the 50% cumulated probability of a Normal distribution:

```
1   %Importance sampling
2   clear all;clc
3   int_Simulations=500;
4   %with significative treshold values no problems for CDF
5   treshold=0;
6   t=1:1:int_Simulations;
7   mu=0;sigma=1;
8   for k=1:1:int_Simulations
9       Nx=normrnd(mu,sigma,1,int_Simulations);
10      cProb(k)=nnz(Nx(:)<=treshold)/int_Simulations;
11  end
12  plot(t,cProb);
13  cProb(k)
14  hy=graph2d.constantline(0.5,'Color',[0.7 0.4 0.6],'LineWidth',2);
15  changedependvar(hy,'y');
```

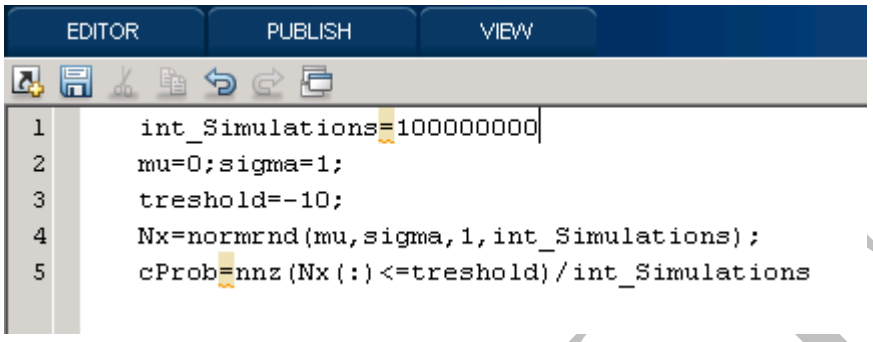
When we run the script we see that if fluctuates for sure but the result quite acceptable:



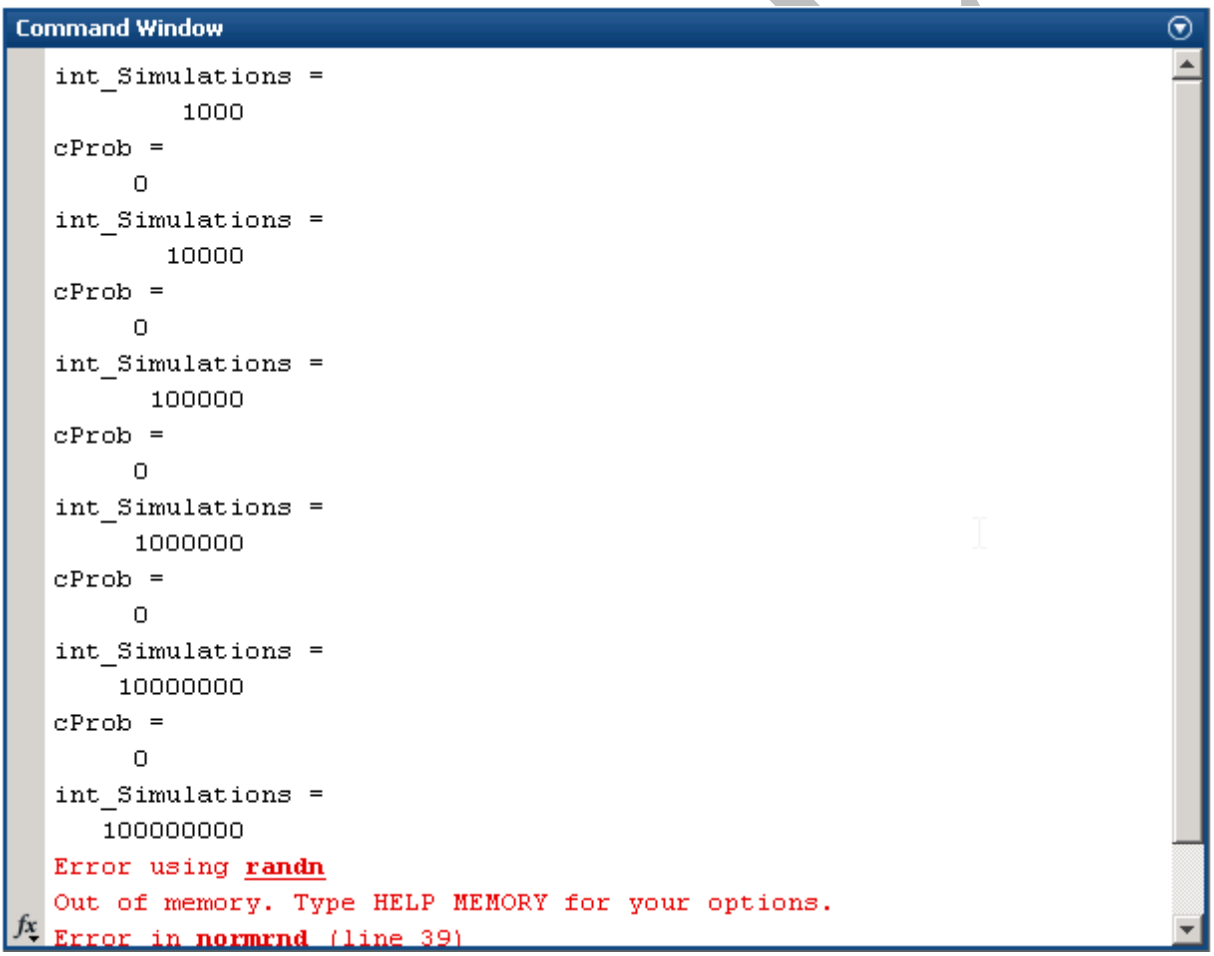
But now… we want to check with MATLAB that:

$$P(X \leq -10) \cong 7.6199 \cdot 10^{-24}$$

and here… we are typically at the tail of the distribution. If we run our script again but without the chart:



We get:



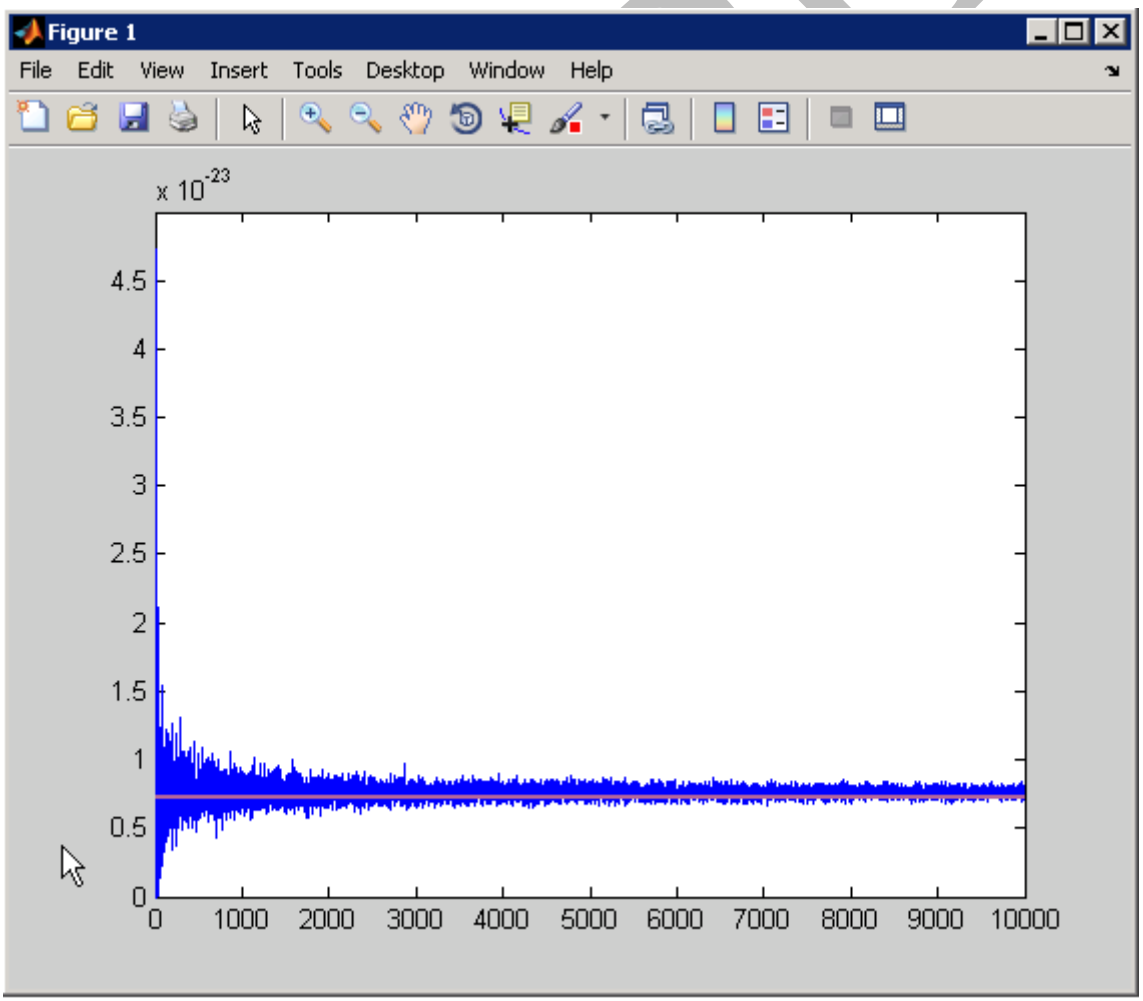
Hummm....

Now we use MC importance sampling technique that we have seen in the theoretical course (the technique is in reality sometimes hard to implement with some distributions because we have to fine the good instrumental distribution):

```
EDITOR  PUBLISH  VIEW

1   int_Simulations=10000;
2   treshold=-10;
3   for k=1:int_Simulations
4       Y=randn(k,1)+treshold;
5       h=(Y<=treshold);
6       m(k)=mean(h.*exp((-treshold)*Y+0.5*(treshold)^2));
7   end
8
9   plot(1:int_Simulations,m)
10  hy=graph2d.constantline(7.1699e-24,'Color',[0.7 0.4 0.6],'LineWidth',2);
11  changedependvar(hy,'y');
```

As we can see the convergence rate has increased considerably!:

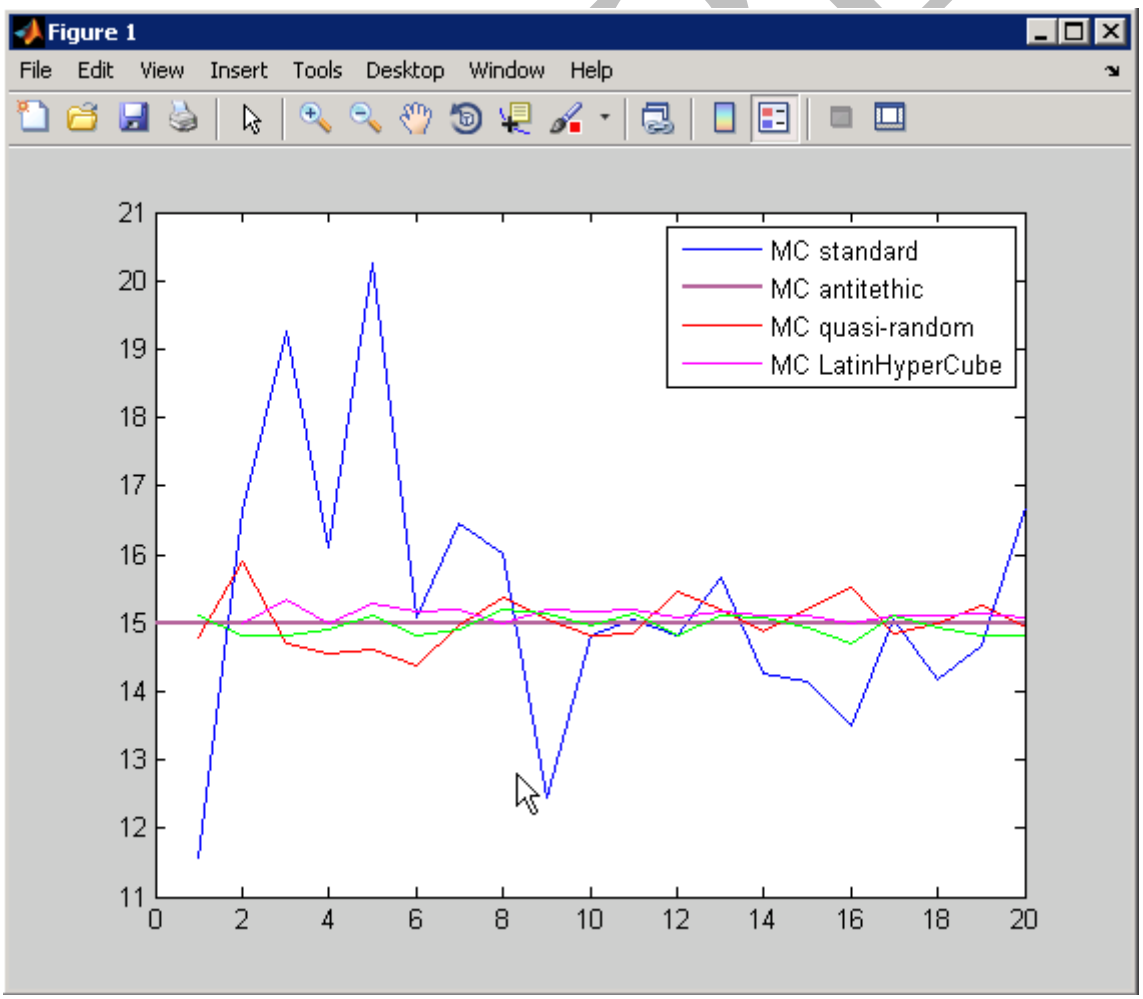


Latin HyperCube

We have also study and applied practically with Microsoft Excel Sheets (it was a nightmare if you remember it...) in the theoretical course a more elaborated method than the previous one called Latin HyperCube. This possibility is implemented in MATLAB trough the function **lhsdesign( )**:

```
40      %MC with LatinHyperCube
41 -    hold on;
42 -    for k=1:1:int_Simulations
43 -        X=lhsdesign(int_Simulations,2);
44 -        x=X(:,1);
45 -        y=X(:,2);
46 -        Nx=norminv(x,mux,xsigma); %N(5,3)
47 -        Ny=norminv(y,muy,ysigma); %N(10,4)
48 -        muNxy(k)=nanmean(Nx+Ny);
49 -    end
50 -    plot(t,muNxy,'g');
51 -    legend('MC standard','MC antitethic','MC quasi-random','MC LatinHyperCube');
```

With 20 simulations we get:

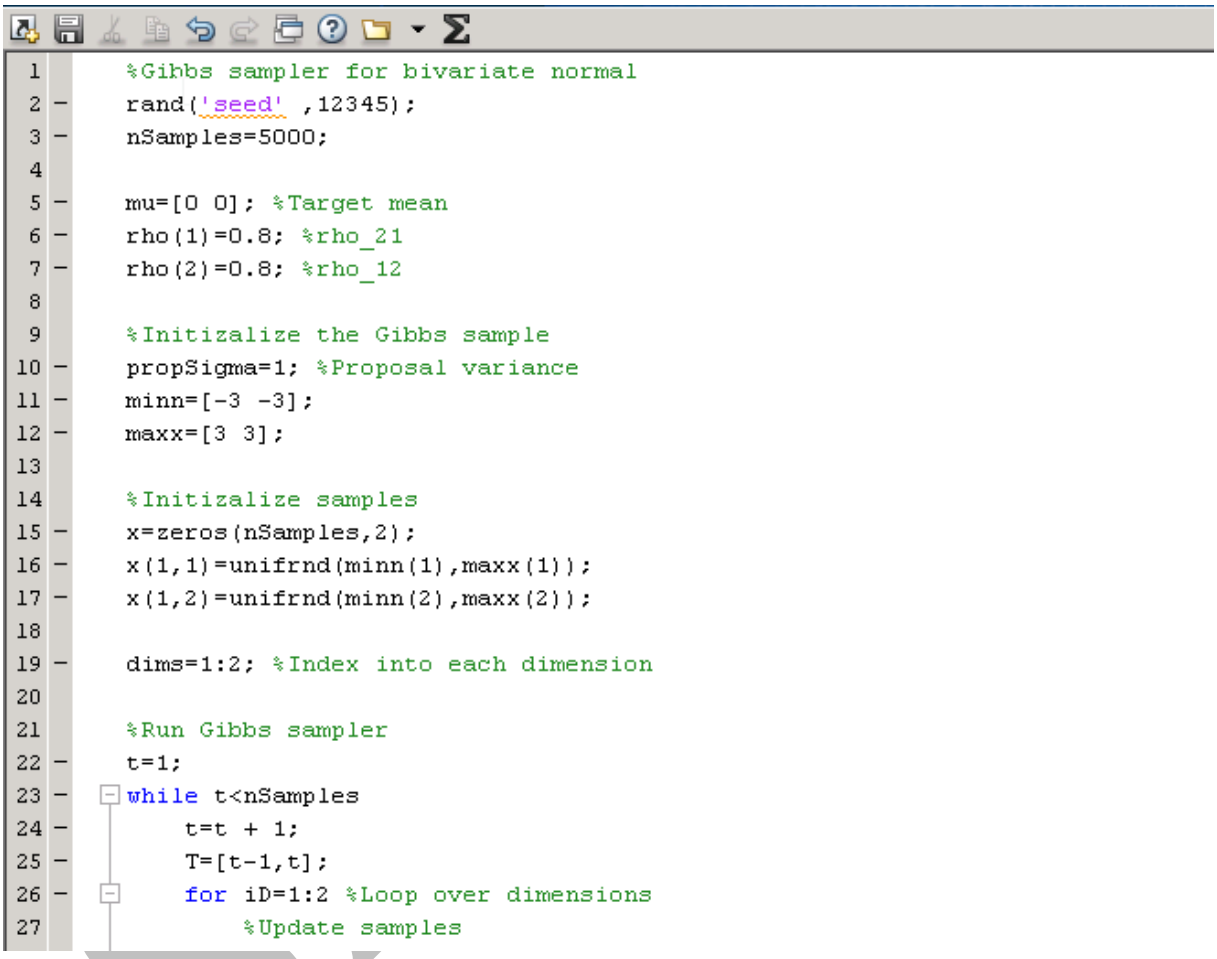


As we can see the difference between the two last methods are qualitatively very small.

Monte Carlo Markov Chain Gibbs Sampling

As we saw it in the theoretical course, Gibbs sampling is a Monte Carlo Markov Chain that helps to generate random variable for multivariate functions in the condition distribution are known in a closed form. We also saw the algorithm converge typically to the joint distribution only if the distribution function is unimodale or has no strong peaks (high leptokurticity).

We will reproduce here the same academic example as the one studied in the theoretical course:



```
%Gibbs sampler for bivariate normal
rand('seed' ,12345);
nSamples=5000;

mu=[0 0]; %Target mean
rho(1)=0.8; %rho_21
rho(2)=0.8; %rho_12

%Initizalize the Gibbs sample
propSigma=1; %Proposal variance
minn=[-3 -3];
maxx=[3 3];

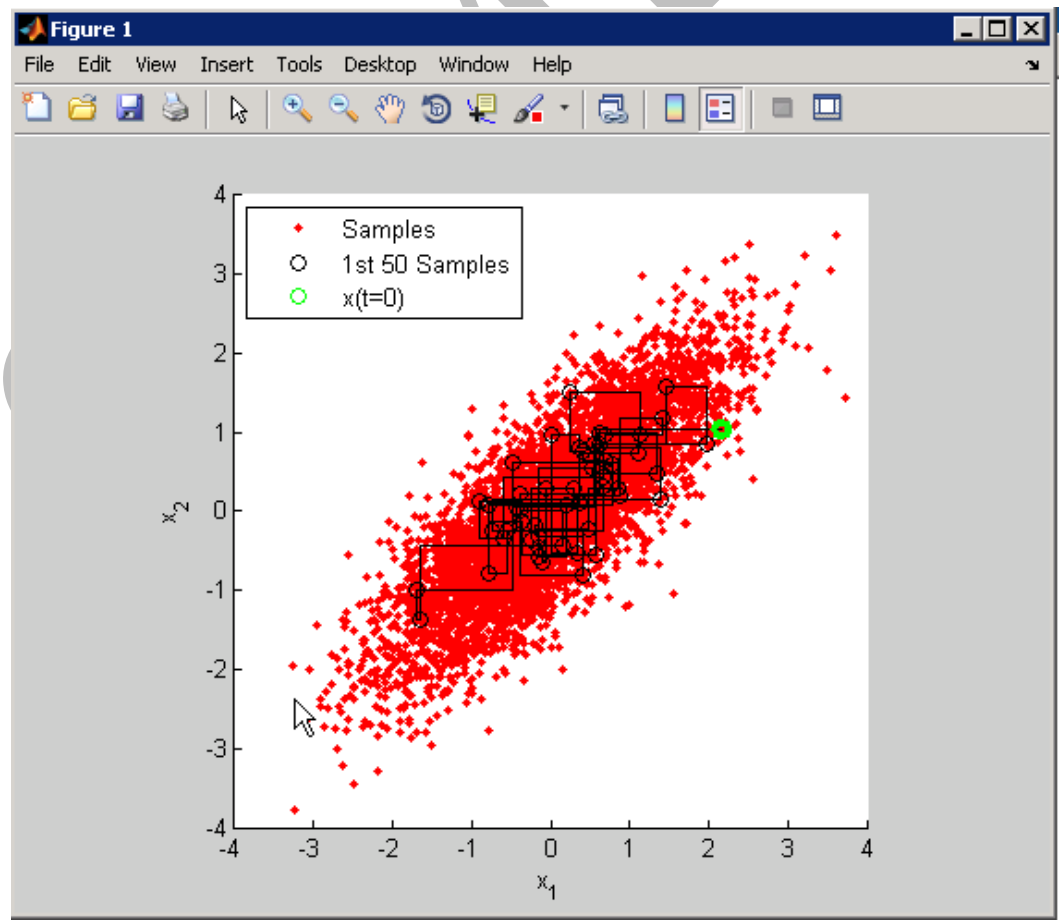
%Initizalize samples
x=zeros(nSamples,2);
x(1,1)=unifrnd(minn(1),maxx(1));
x(1,2)=unifrnd(minn(2),maxx(2));

dims=1:2; %Index into each dimension

%Run Gibbs sampler
t=1;
while t<nSamples
    t=t + 1;
    T=[t-1,t];
    for iD=1:2 %Loop over dimensions
        %Update samples
```

```
28 -            nIx=dims~=iD; %*NOT* the current dimension
29              %Conditional mean
30 -            muCond=mu(iD) + rho(iD)*(x(T(iD),nIx)-mu(nIx));
31              %Conditiona variance
32 -            varCond=sqrt(1-rho(iD)^2);
33              %Draw form conditional
34 -            x(t,iD)=normrnd(muCond,varCond);
35 -        end
36 -    end
37
38      %Display sampling dynamics
39 -    figure;
40 -    h1 = scatter(x(:,1),x(:,2),'r.');
41
42      %Conditional step/samples
43 -    hold on;
44 -    for t = 1:50
45 -        plot([x(t,1),x(t+1,1)],[x(t,2),x(t,2)],'k-');
46 -        plot([x(t+1,1),x(t+1,1)],[x(t,2),x(t+1,2)],'k-');
47 -        h2 = plot(x(t+1,1),x(t+1,2),'ko');
48 -    end
49
50 -    h3 = scatter(x(1,1),x(1,2),'go','Linewidth',3);
51 -    legend([h1,h2,h3],{'Samples','1st 50 Samples','x(t=0)'},'Location','Northwest')
52 -    hold off;
53 -    xlabel('x_1');ylabel('x_2');axis square
```

If we run it we get:



Optimization (with Optimization Toolbox/Global Optimization Toolbox)

Here we will try to reproduce all examples we made with the Microsoft Excel solver excepted those related to finance that are reserved to the finance chapter (se page 983). For sure we will also tried to reproduce also the few optimization problems that we solved by hand in the theoretical course!

http://www.MathWorks.ch/ch/help/optim/release-notes.html

http://www.MathWorks.ch/ch/help/gads/release-notes.html

Linear optimization using LP Simplex (LPS)

All examples are taken from the theoretical training on Numerical Methods. I tried to order the examples in ascending order of complexity:

LP-S Factory optimization problem (inequality linear system)

This is the easiest optimization problem that we solved by hand in the theoretical course using the simplex algorithm. We use for this purpose first the function **optimoptions( )** to define the parameters of the algorithm and finally the function **linprog( )**.

We must not forget that as given by MATHWORKS the syntax of **linprog( )** is:

Finds the minimum of a problem specified by

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

SAMPLE

And the problem we solve in the theoretical course was (factory allocation problem):

$$3x_1 + 4x_2 = 160$$
$$6x_1 + 3x_2 = 180$$
$$x_1, x_2 \geq 0$$
$$\max(Z) = 1200x_1 + 1000x_2$$

Maximizing Z corresponds to minimizing the same equation with negative coefficients.

EDITOR PUBLISH VIEW

```
1   %we force the use of the simplex algorithm
2   clear all;clc
3   options=optimoptions('linprog','Algorithm','simplex');
4   %by default MATLAB minimize then we have to put a "-" in front
5   %of the constants of the economic function
6   f=[-1200 -1000];
7   A=[3 4;6 3];
8   b=[160 180];
9   lb=zeros(2,1);
10  [x,fval,exitflag,output,lambda]=linprog(f,A,b,[],[],lb,[],[],options)
```

SAMPLE

And we get exactly the same results as those calculated by hand:

```
Command Window
Optimization terminated.

x =

    16
    28


fval =

      -47200


exitflag =

     1


output =

         iterations: 2
          algorithm: 'simplex'
      cgiterations: []
            message: 'Optimization terminated.'
    constrviolation: 0
      firstorderopt: 0

lambda =

    ineqlin: [2x1 double]
      eqlin: [0x1 double]
      upper: [2x1 double]
      lower: [2x1 double]
```

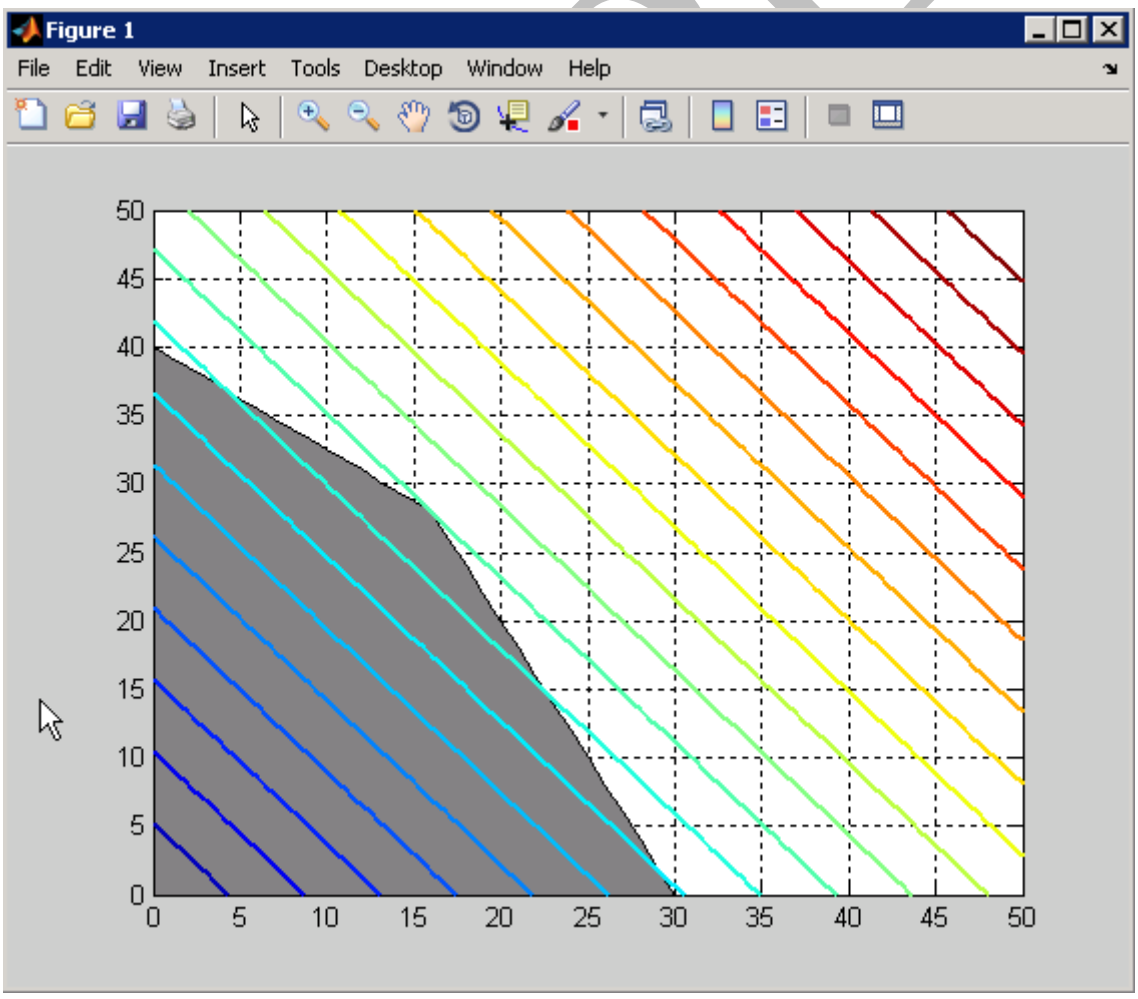
If you use the *exitflag* parameter and define it as equal to -3 then the solver will not be able to found a feasible solution!

As a bonus let us plot the famous corresponding polygon constraints chart that is not really intuitive to build:

EDITOR PUBLISH VIEW

```
%not intuitive at all the build the polygon chart of constraints...
x=0:50; % range for chart
y1=max((180-6*x)/3,0);
y2=max((160-3*x)/4,0);
y3=max((180-6*x)/3,0);
ytop=min([y1;y2;y3]);
area(x,ytop,'FaceColor',[0.5 0.5 0.5]);

hold on;
[u, v]=meshgrid(0:50,0:50);
[c, h]=contour(u,v,1200*u+1000*v,20);
set(h,'linewidth',2);
grid on;
```



LP-S Naive (without covariance…) Markowitz parametric portfolio optimization problem (equality linear system without target value)

This is another very naïve (and even stupid) example of the use onf linprog for a portfolio optimization without taking into account the covariance matrix. The interesting thing in this example here is that we use different arguments relatively to the previous example because the situation is simply different!

Once again remember that:

Finds the minimum of a problem specified by

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

And what we want it is the naïve problem formally used in the theoretical course:

$$\max(Z) = 0.116w_1 + 0.226w_2 + 0.252w_3$$

$$\sum_i w_i = 1$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \le \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \le \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

SAMPLE

This will be written as:

```
clear all;clc
v_Ret=[0.116 0.226 0.252];

int_Assets=length(v_Ret);
lb=zeros(1,int_Assets);
ub=ones(1,int_Assets);
Aeq=[ones(1,int_Assets)];
Beq=[1];

%very naive first example...
%without taking the covariance matrix in consideration... O_o
options=optimoptions('linprog','Algorithm','simplex')
[TargetReturnWeights, ~, exitflag]=linprog(-v_Ret,[],[],Aeq,Beq,lb,ub,[],options)
Return=v_Ret*TargetReturnWeights
```

That gives:

```
Optimization terminated.

TargetReturnWeights =

     0
     0
     1


exitflag =

     1


Return =

    0.2520

>>
```

This is naively the trivial result!

LP-S Naive (without covariance…) Markowitz parametric portfolio optimization problem (equality linear system with target value)

Here we complexify the previous example as in the theoretical course:

Once again remember that:

Finds the minimum of a problem specified by

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

And what we want now is formally:

$$0.116w_1 + 0.226w_2 + 0.252w_3 = 0.2$$

$$\sum_i w_i = 1$$

$$\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \le \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \le \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

The weights can be negative for short positions.

As you can see here, we have a function with a targeted strict equality value! Because the covariance is still not taking into account (for the moment) we can still use the simplex method.

SAMPLE

With MATLAB syntax this will be written:

```
clear all;clc
v_Ret=[0.116 0.226 0.252];

int_Assets=length(v_Ret);
lb=zeros(1,int_Assets);
ub=ones(1,int_Assets);
TargetReturn=0.20; %when greater than 0.252 they will be logically no feasible solution
Aeq=[v_Ret;ones(1,int_Assets)];
Beq=[TargetReturn;1];

%very naive first example...
%without taking the covariance matrix in consideration... O_o
options=optimoptions('linprog','Algorithm','simplex')
[TargetReturnWeights, ~, exitflag]=linprog(-v_Ret,[],[],Aeq,Beq,lb,ub,[],options)
Return=v_Ret*TargetReturnWeights
```

So we get as output:

```
Optimization terminated.

TargetReturnWeights =

    0.3824
         0
    0.6176


exitflag =

     1


Return =

    0.2000
```

And this is one possible correct result.

More interesting version including covariance will come in the next example.

Function constrained optimization (FCO)

Now the problems we will start to solve a nonlinear anymore. We will still use the examples studied in the theoretical course.

FCO with constraints and without specific target

We will use now the function **fmincon( )** that means "find minimum including constraints" to find the minimum global variance Markowitz portfolio without fixing explicitly a given return.

The MathWorks says us that:

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \le 0 \\ ceq(x) = 0 \\ A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub, \end{cases}$$

b and *beq* are vectors, *A* and *Aeq* are matrices, *c*(*x*) and *ceq*(*x*) are functions that return vectors, and *f*(*x*) is a function that returns a scalar. *f*(*x*), *c*(*x*), and *ceq*(*x*) can be nonlinear functions.

x, *lb*, and *ub* can be passed as vectors or matrices; see Matrix Arguments.

Syntax

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
```

And our problem is formally written as seen in the theoretical course:

$$\min(Z) = \min(V(R)) = \vec{w}^T \begin{bmatrix} 0.21728 & -0.003376 & -0.053492 \\ -0.003376 & 0.00253 & 0.008468 \\ -0.053492 & 0.008468 & 0.2247 \end{bmatrix} \vec{w}$$

$$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot \vec{w} = 1$$

$$\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \le \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \le \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

With initial numerical values to found a solution:

$$\vec{w}_{0,i} = \vec{0}$$

SAMPLE

In MATLAB language script this will be written:

```
%Markowitz global minimum variance portfolio
clear all;close all;clc
v_Ret=[0.116 0.226 0.252];
m_Cov=[0.21728 -0.003376 -0.053492;
        -0.003376 0.00253 0.008468
        -0.053492 0.008468 0.2247
        ];

TargetReturn=0.22;
X0=zeros(3,1);
Aeq=ones(1,3);
Beq=1;
lb=-ones(3,1);
ub=ones(3,1);

Variance=@(w)w'*m_Cov*w; %analytical function expression of the variance!!!
[OptimalWeights, OptimalVariance]=fmincon(Variance,X0,[],[],Aeq,Beq,lb,ub)

Return=v_Ret*OptimalWeights %check tarsuccessful
```

That gives us the expected results:

```
Command Window
is less than the default value of the function tolerance and constra
are satisfied to within the default value of the constraint toleranc

<stopping criteria details>

Active inequalities (to within options.TolCon = 1e-06):
  lower      upper     ineqlin   ineqnonlin
                2

OptimalWeights =

    0.0216
    1.0000
   -0.0216


OptimalVariance =

    0.0023


Return =

    0.2231

fx >>
```

FCO with constraints and specific target

We will use now the function **fmincon( )** that means "find minimum including constrains" to find the minimum global variance Markowitz portfolio by fixing also explicitly a given return.

The MathWorks says us that:

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \le 0 \\ ceq(x) = 0 \\ A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub, \end{cases}$$

b and *beq* are vectors, *A* and *Aeq* are matrices, *c*(*x*) and *ceq*(*x*) are functions that return vectors, and *f*(*x*) is a function that returns a scalar. *f*(*x*), *c*(*x*), and *ceq*(*x*) can be nonlinear functions.

x, *lb*, and *ub* can be passed as vectors or matrices; see Matrix Arguments.

Syntax

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
```

And our problem is formally written as seen in the theoretical course:

$$0.116w_1 + 0.226w_2 + 0.252w_3 = 0.22$$

$$\min\left(Z\right) = \min(V\left(R\right)) = \vec{w}^T \begin{bmatrix} 0.21728 & -0.003376 & -0.053492 \\ -0.003376 & 0.00253 & 0.008468 \\ -0.053492 & 0.008468 & 0.2247 \end{bmatrix} \vec{w}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0.116 & 0.226 & 0.252 \end{bmatrix} \cdot \vec{w} = \begin{pmatrix} 1 \\ 0.22 \end{pmatrix}$$

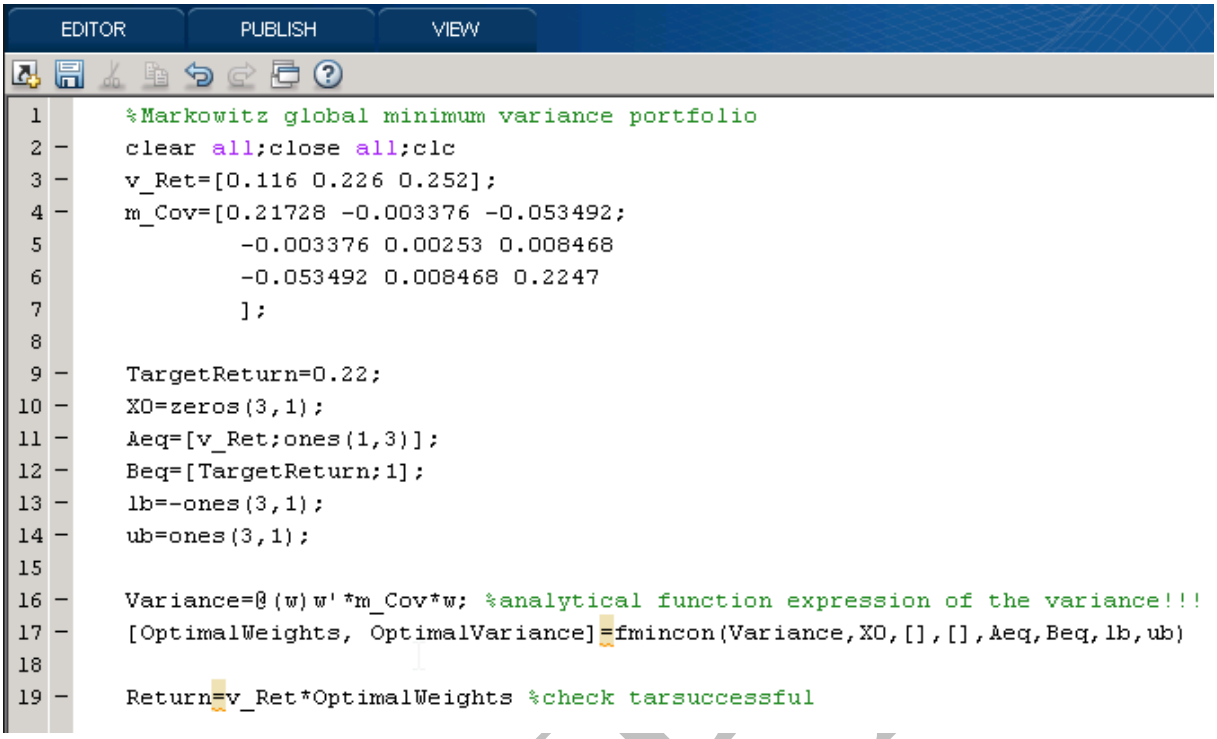
$$\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \le \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \le \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

With initial numerical values to found a solution:

$$\vec{w}_{0,i} = \vec{0}$$

SAMPLE

In MATLAB language script this will be written:




```
%Markowitz global minimum variance portfolio
clear all;close all;clc
v_Ret=[0.116 0.226 0.252];
m_Cov=[0.21728 -0.003376 -0.053492;
        -0.003376 0.00253 0.008468
        -0.053492 0.008468 0.2247
        ];

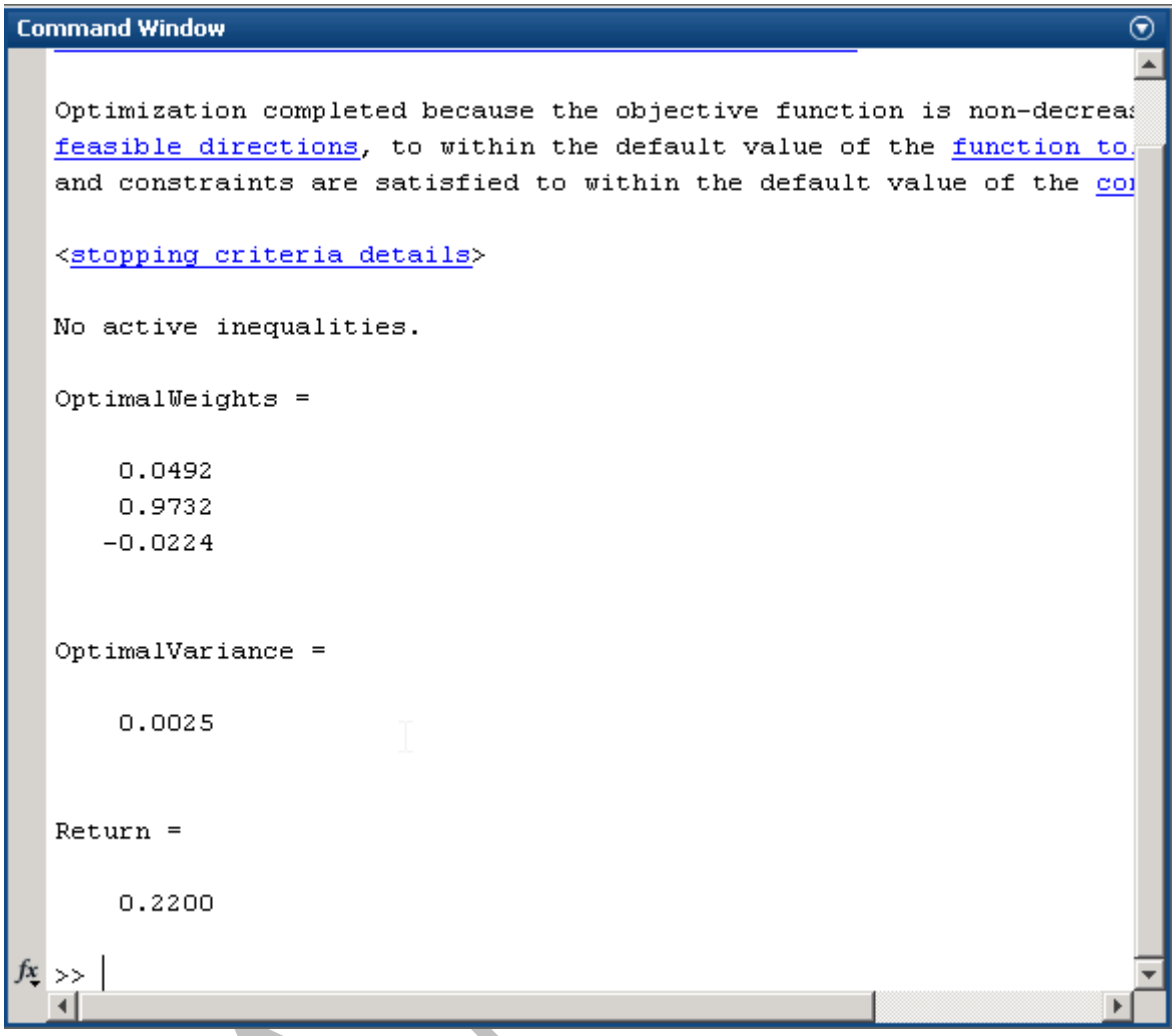
TargetReturn=0.22;
X0=zeros(3,1);
Aeq=[v_Ret;ones(1,3)];
Beq=[TargetReturn;1];
lb=-ones(3,1);
ub=ones(3,1);

Variance=@(w)w'*m_Cov*w; %analytical function expression of the variance!!!
[OptimalWeights, OptimalVariance]=fmincon(Variance,X0,[],[],Aeq,Beq,lb,ub)

Return=v_Ret*OptimalWeights %check tarsuccessful
```

SAMPLE

That gives us the expected results:



FCO with nonlinear constraints (and specific target)

An academic classical and trivial example of a FCO with nonlinear constraints and specific target is the shipping box problem. We consider a parallelepiped with the following information:

$$V = x \cdot y \cdot z = 8$$
$$\min(Z) = \min(S) = 2\left(xy + xz + yz\right)$$
$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} +\infty \\ +\infty \\ +\infty \end{pmatrix}$$

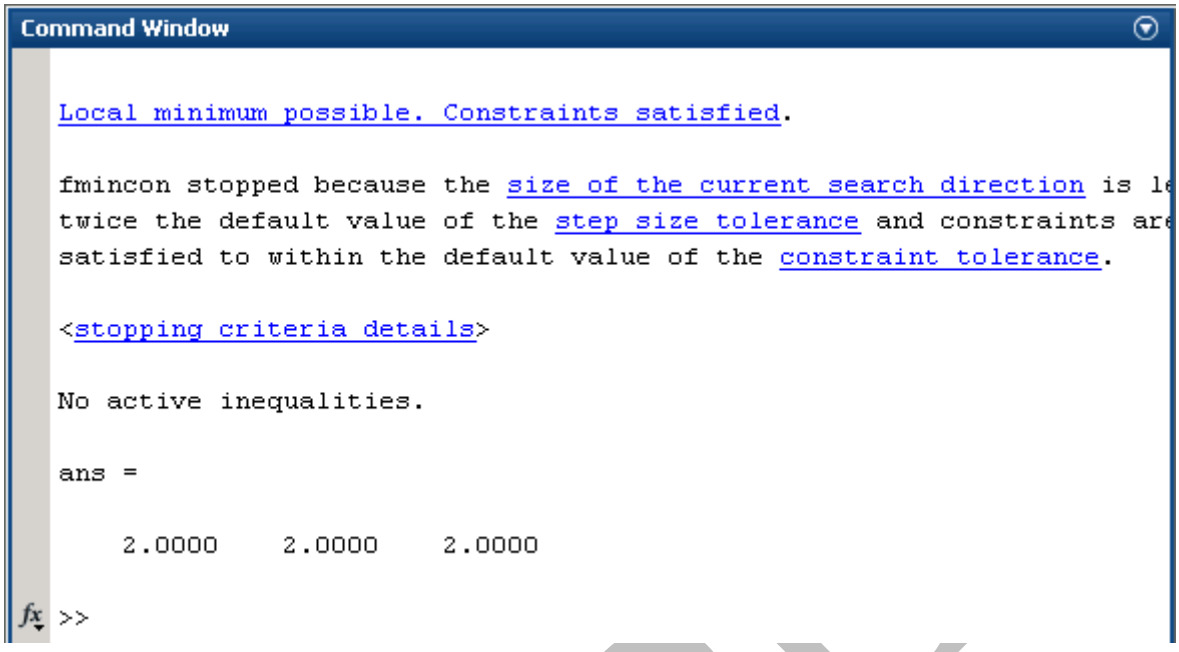
In MATLAB language script this will be written in the non-trivial following form:

```
EDITOR  PUBLISH  VIEW
1  %shipping box problem:
2  %found the dimensions of a box that for a given volume V=x*y*z=8
3  %minimize the total surface 2*(x*y+x*z+y*z)
4  warning('off','all'); %we hide the warnings of MATLAB
5  x0=[1 1 1];
6  lb=[0 0 0];
7  up=[Inf Inf Inf];
8  fmincon(inline('2*(x(1)*x(2)+x(1)*x(3)+x(2)*x(3))'),x0,[],[],[],[],lb,up,@nlcon)
```

With the external nonlinear function (nlcon):

```
C:\Documents and Settings\Administrateur\Mes documents\MATLAB\nlcon.m
EDITOR  PUBLISH  VIEW
1  function [C,Ceq]=nlcon(x)
2      C=[];
3      Ceq=[x(1)*x(2)*x(3)-8];
4  end
```

We get as result the expected trivial answer (cube):



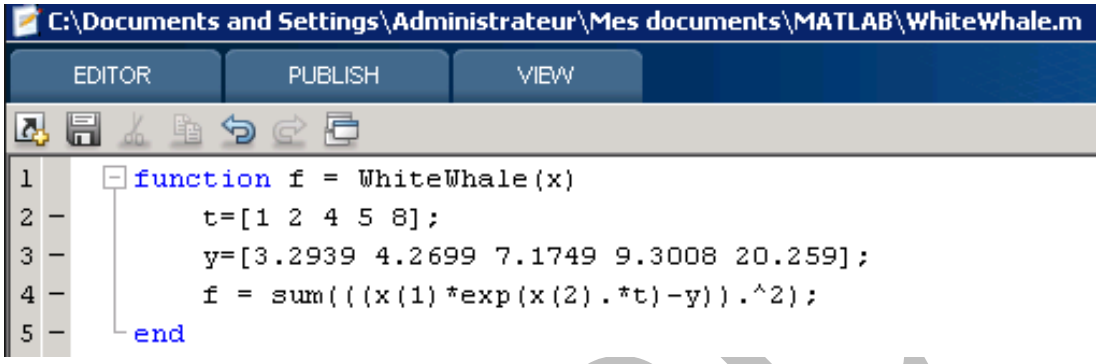
SAMPL

Quadratic optimization (QO)

QO using conjugate gradient method (Q-CG)

This is also called "non-constrained" or "unconstrained" optimization problem. We will use the same function as in the theoretical course.

First we define our analytical function expression we our known points:



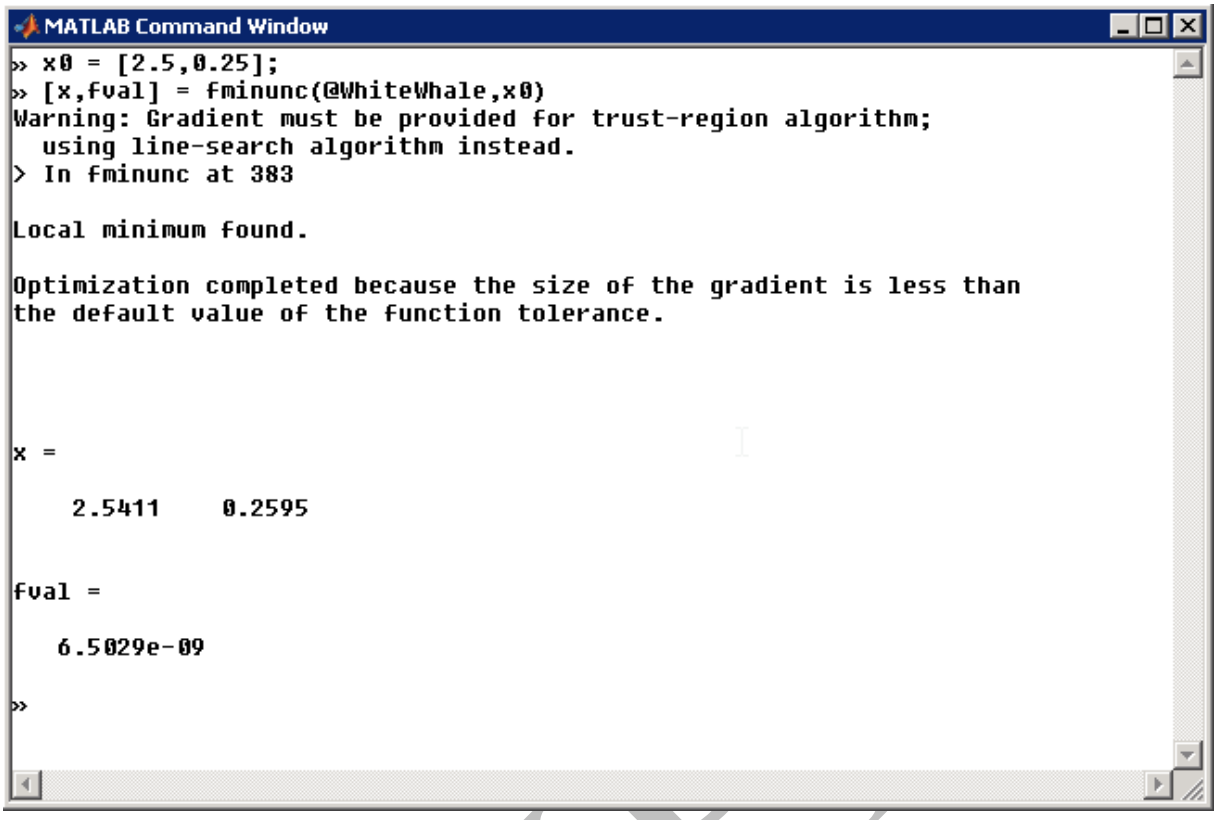
As we search *a* and *b* that minimize the sum of square error:

$$\sum_i \left(ae^{bt_i} - y_i \right)^2$$

One this done, we use[4] **fminunc( )** where is for "unc"-constrained at the opposite of **fmincon( )** we saw before:

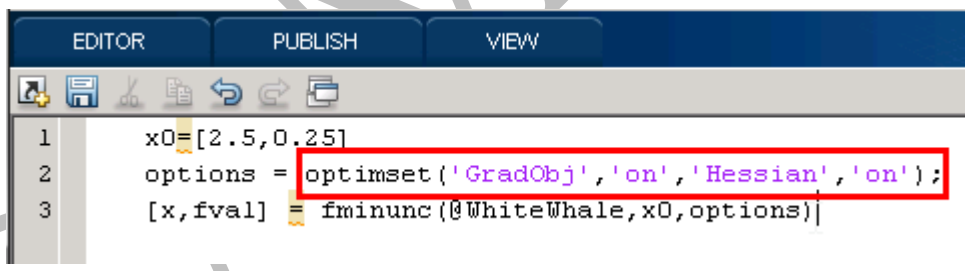
[4] In the facts fminunc( ) can solve not only quadratic optimization problems but this is just an example to show that we can also use this function for such problems.

SAMPL

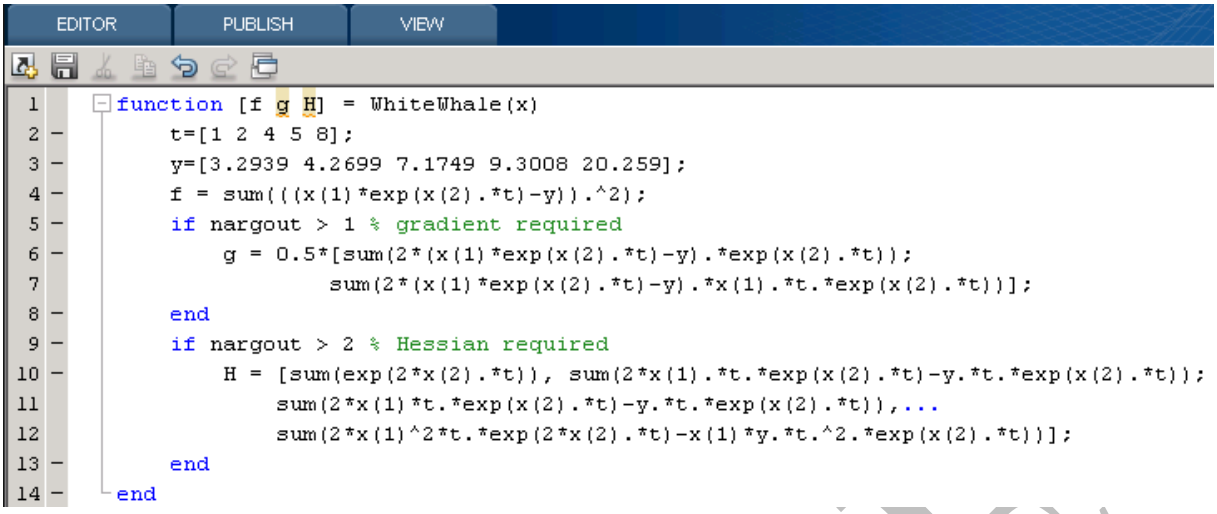


We do not get exactly the same result as the one calculated by hand in the theoretical course.

We can rewrite **fminunc( )** be expliciting the Hessian and Gradient matrix the we have calculated by hand in the theoretical course:



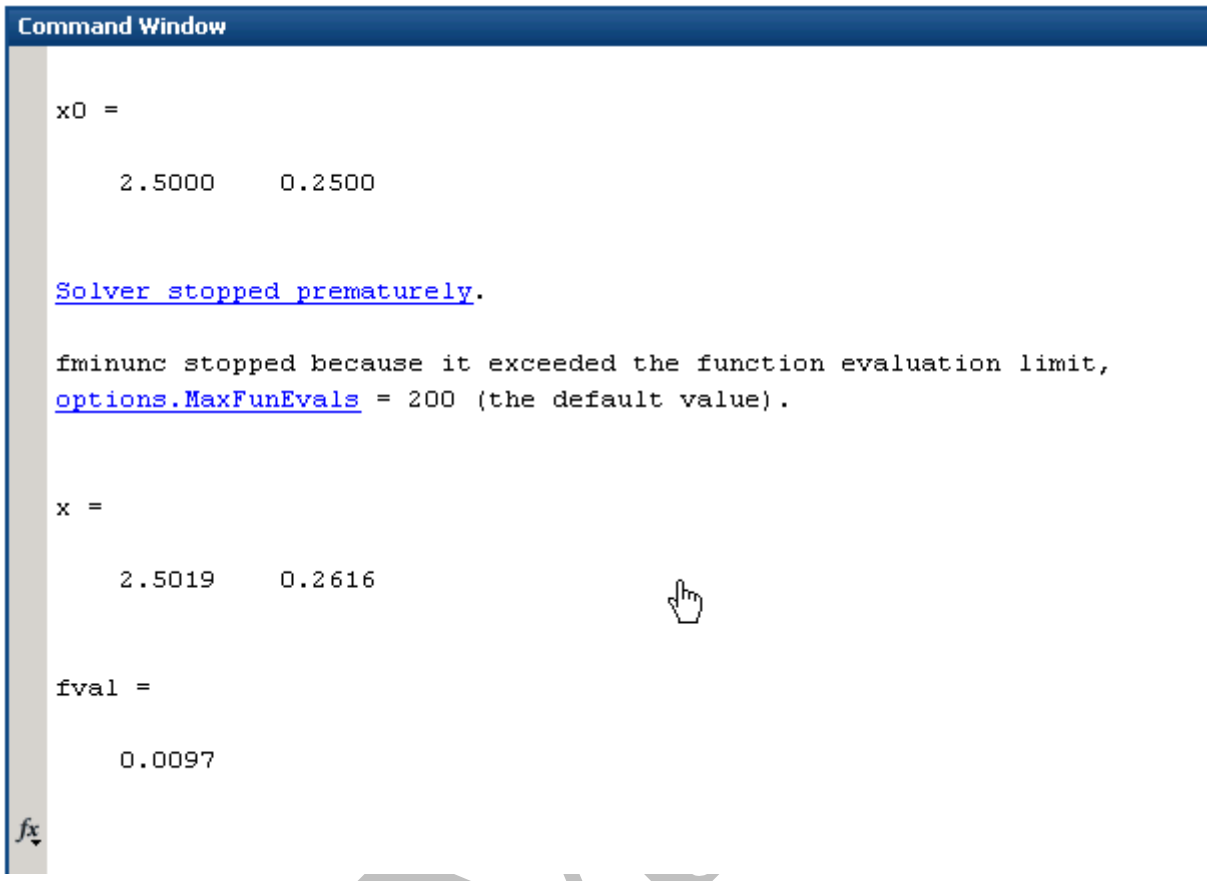
This gives:



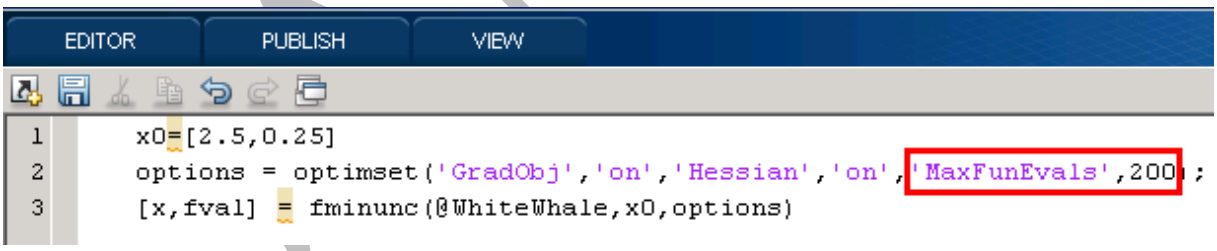

```
function [f g H] = WhiteWhale(x)
    t=[1 2 4 5 8];
    y=[3.2939 4.2699 7.1749 9.3008 20.259];
    f = sum(((x(1)*exp(x(2).*t)-y)).^2);
    if nargout > 1 % gradient required
        g = 0.5*[sum(2*(x(1)*exp(x(2).*t)-y).*exp(x(2).*t));
                 sum(2*(x(1)*exp(x(2).*t)-y).*x(1).*t.*exp(x(2).*t))];
    end
    if nargout > 2 % Hessian required
        H = [sum(exp(2*x(2).*t)), sum(2*x(1).*t.*exp(x(2).*t)-y.*t.*exp(x(2).*t));
            sum(2*x(1)*t.*exp(x(2).*t)-y.*t.*exp(x(2).*t)),...
            sum(2*x(1)^2*t.*exp(2*x(2).*t)-x(1)*y.*t.^2.*exp(x(2).*t))];
    end
end
```

SAMPLE

And we can see that the result is worse than before. Because MATLAB did 200 hundred iterations we cannot compare with the few iterations we calculated by hand:



For sure we can control the number of iterations with MATLAB but in practice we almost always take the maximum and not just three iterations…:



QO using interior point convex method (Q-IPC)

To study a specialized optimization function of MATLAB for quadratic problems we will take the Markowitz global minimum variance problem again first without target return value.

First MathWorks give us that:

Finds a minimum for a problem specified by

$$\min_x \frac{1}{2}x^T Hx + f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

H, *A*, and *Aeq* are matrices, and *f*, *b*, *beq*, *lb*, *ub*, and *x* are vectors.

```
x = quadprog(H,f)
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
x = quadprog(problem)
```

Our Markowitz problem is formally:

$$\min(Z) = \min(V(R)) = \vec{w}^T \begin{bmatrix} 0.21728 & -0.003376 & -0.053492 \\ -0.003376 & 0.00253 & 0.008468 \\ -0.053492 & 0.008468 & 0.2247 \end{bmatrix} \vec{w}$$

$$\begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \le \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \le \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

SAMPLE

In MATLAB language script this will be written:

```
clear all;clc
v_Ret=[0.116 0.226 0.252];
m_Cov=[0.21728 -0.003376 -0.053492;
      -0.003376 0.00253 0.008468;
      -0.053492 0.008468 0.2247];
int_Assets=length(v_Ret);
lb=-ones(1,int_Assets);
ub=ones(1,int_Assets);
Aeq=ones(1,int_Assets);
sng_SumOfWeightsConstraint=1;
MinVarWeights=quadprog(m_Cov,[0 0 0],[],[],Aeq,1,lb,ub);
MinVarWeights
MinVarStd=sqrt(MinVarWeights'*m_Cov*MinVarWeights);
MinStd=MinVarStd^(1/2)
MaxReturn=v_Ret*MinVarWeights
```

As we can see it is normal than we get exactly the same result as the one we get on page 961:

```
Optimization terminated.

MinVarWeights =

    0.0216
    1.0000
   -0.0216


MinStd =

    0.2184


MaxReturn =

    0.2231
```

And the same with a target return now:

```
clear all;clc
v_Ret=[0.116 0.226 0.252];
m_Cov=[0.21728 -0.003376 -0.053492;
       -0.003376 0.00253 0.008468;
       -0.053492 0.008468 0.2247];
int_Assets=length(v_Ret);
lb=-ones(1,int_Assets);
ub=ones(1,int_Assets);
Aeq=ones(1,int_Assets);
sng_SumOfWeightsConstraint=1;
MinVarWeights=quadprog(m_Cov,[0 0 0],v_Ret,0.22,Aeq,1,lb,ub);
MinVarWeights
MinVarStd=sqrt(MinVarWeights'*m_Cov*MinVarWeights);
MinStd=MinVarStd^(1/2)
MaxReturn=v_Ret*MinVarWeights
```

This gives us exactly the same result as the one of page 963:

```
Optimization terminated.

MinVarWeights =

    0.0492
    0.9732
   -0.0224


MinStd =

    0.2227


MaxReturn =

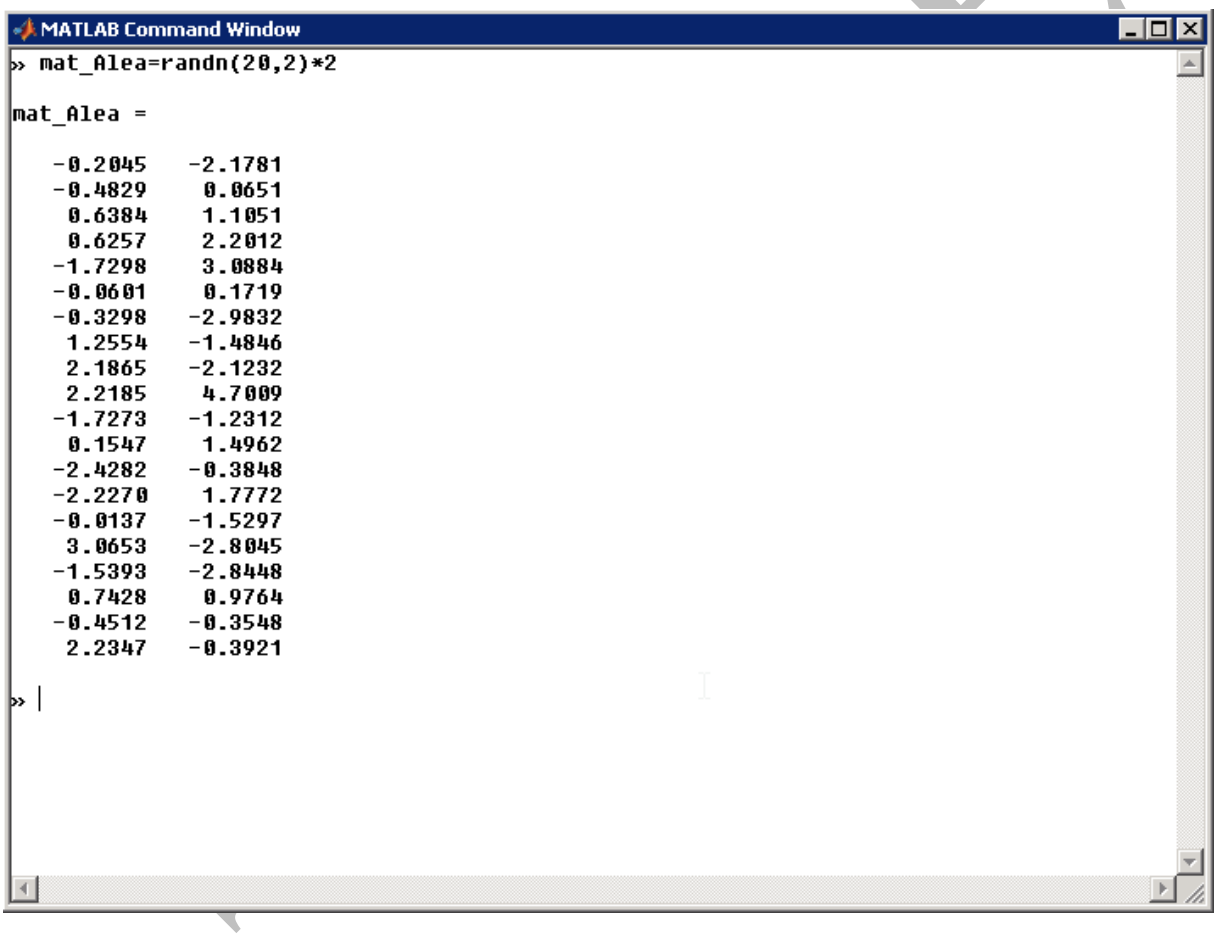
    0.2200
```

Genetic algorithms (GA)

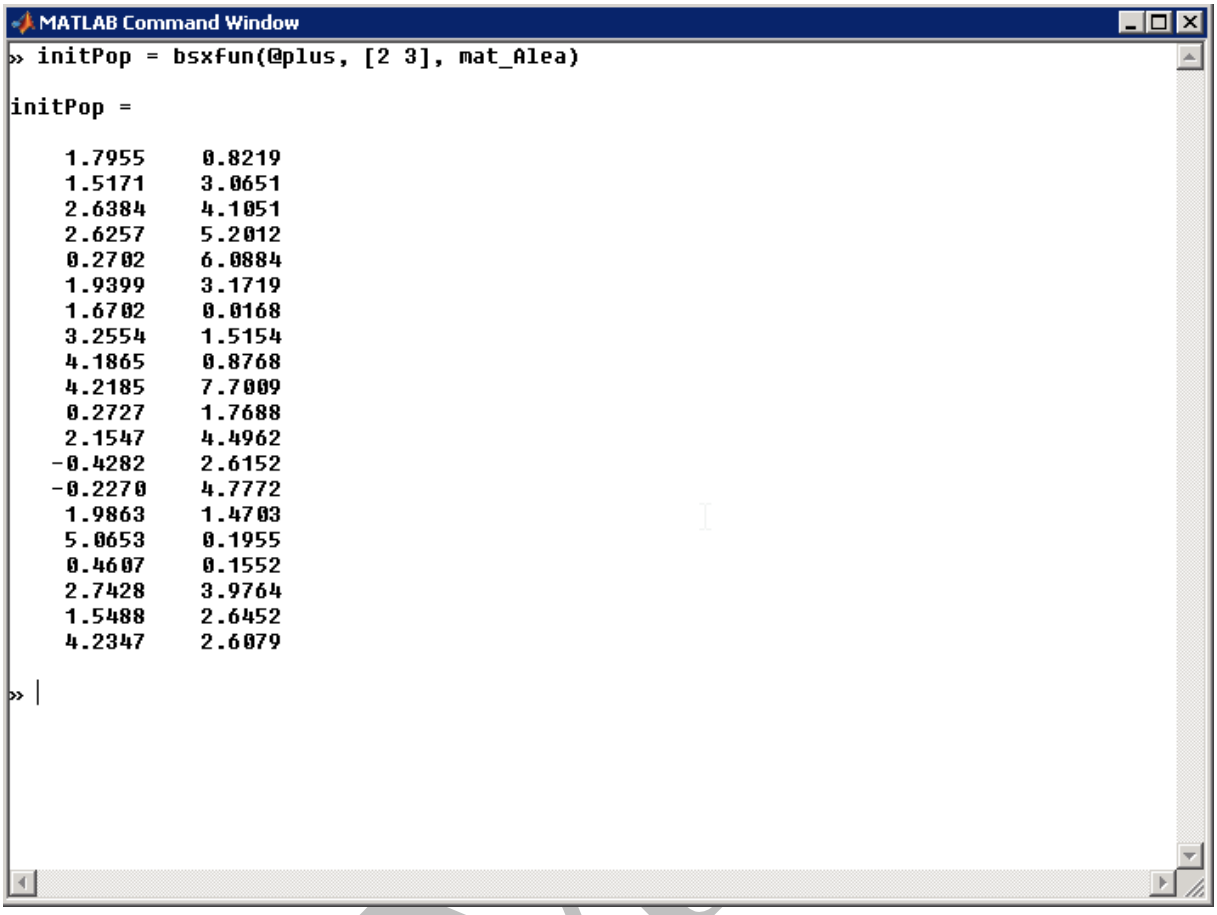
We have studied in the theoretical course of a Numerical Methods the idea behind genetic algorithms and we also play with such an algorithm with the new solver available since Microsoft Excel 2010. The idea here is to see how the same optimization tool works in MATLAB and after try to reproduce the same results as the one we did in Microsoft Excel.

Simple close-form unconstrained function without upper and lower bounds

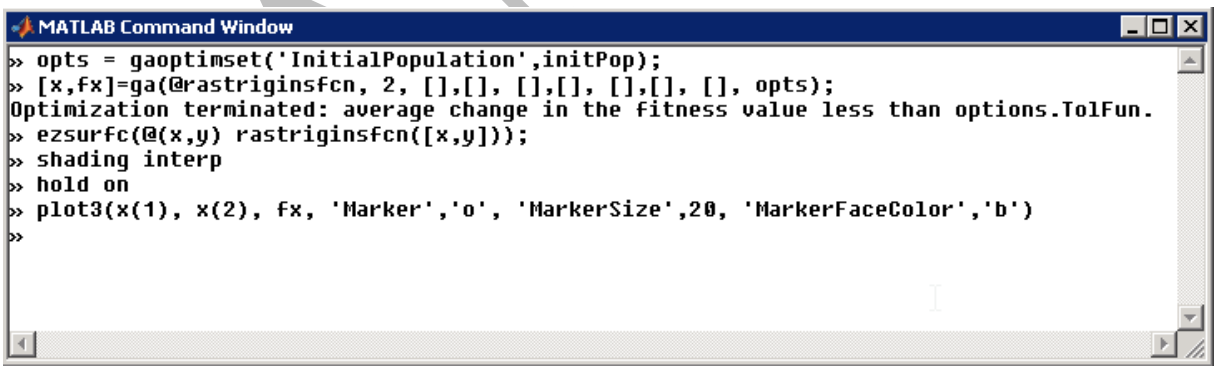
The first classical academic example of GA is to find the optimum of Rastriging functions. For this purpose we first build a 2D random bounded grid as initial population for the evolution:



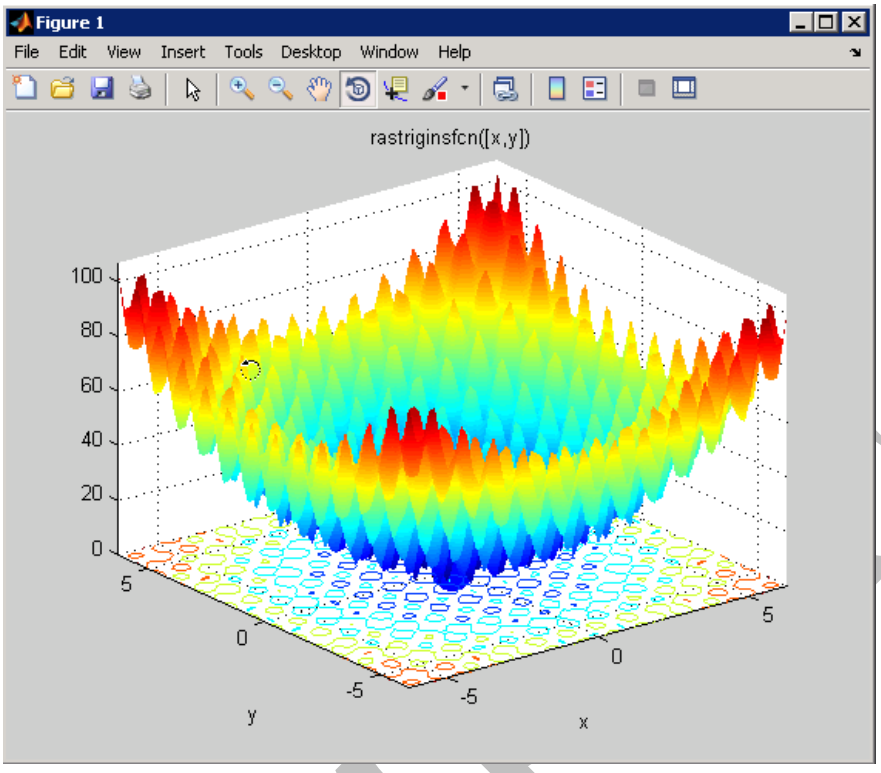
We now shift the values because we now the 2D definition domain of the Rastriging function:



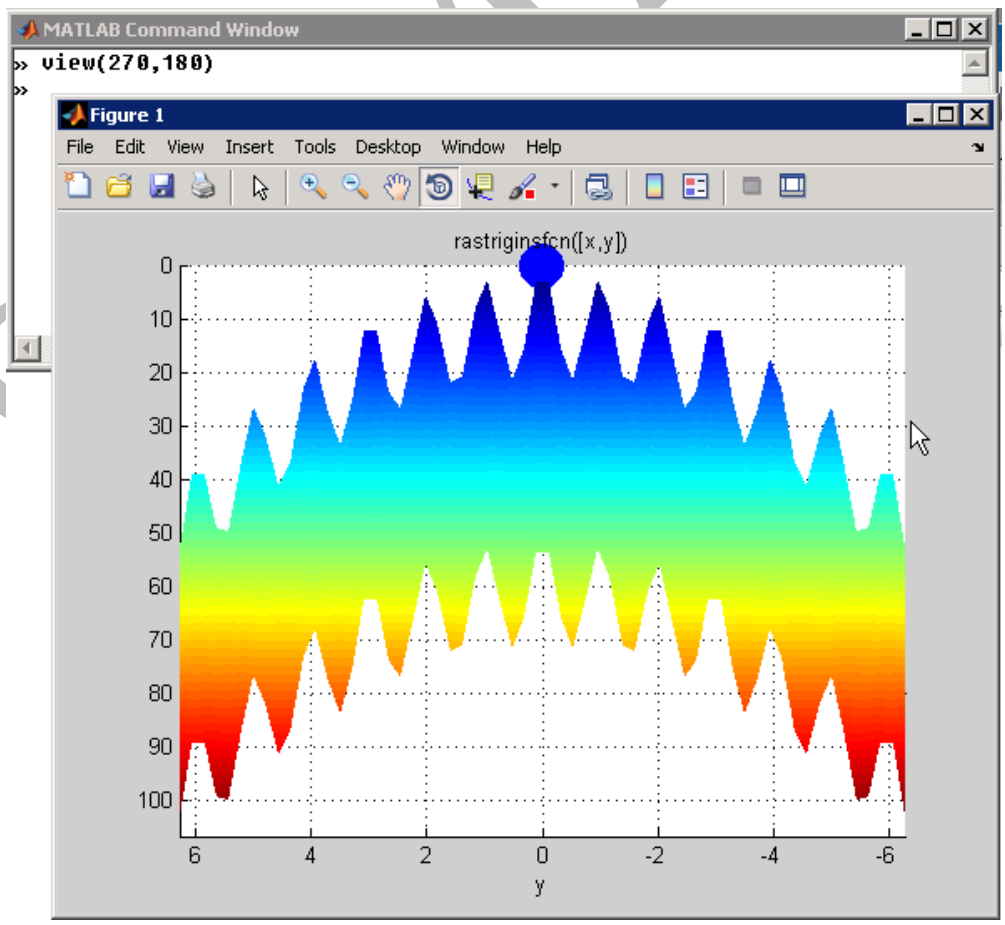
Now we use the function **gaoptimset( )** function to define the initial population to use and after the **ga( )** function to search the optimum and plot the solution:



So we get:



If we change a little bit the perspective:

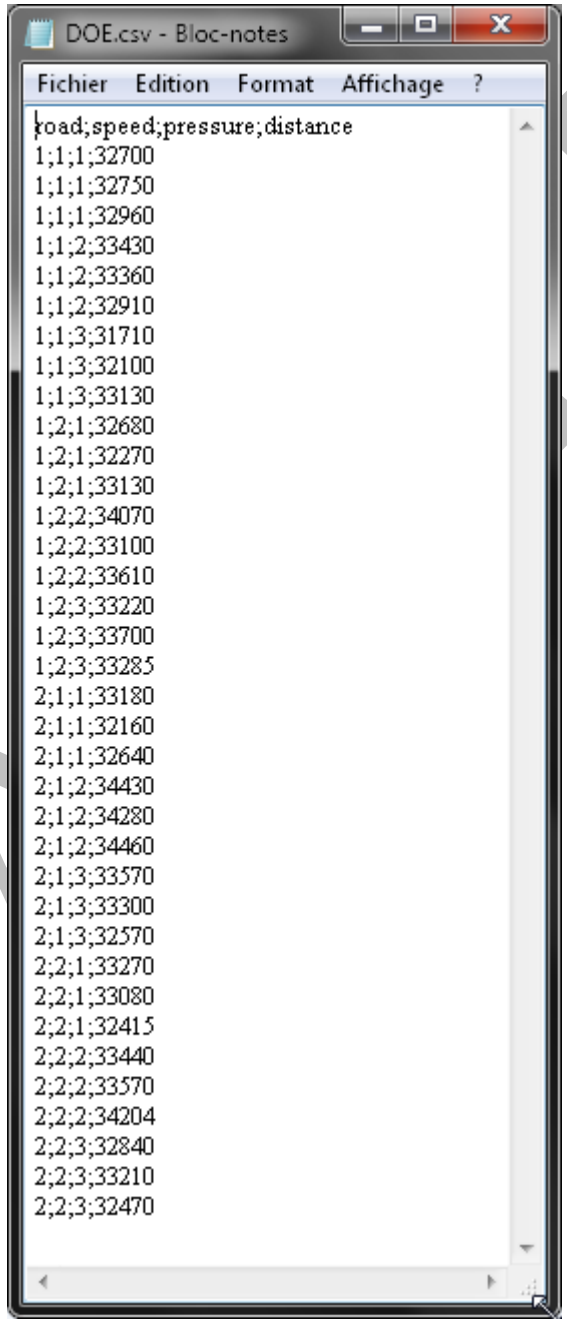


Neural Network GA optimization for DOE

In this example we will use two thing for which he have study the idea in the theoretical course of Numerical Methods: 1) The genetic algorithms (GA), 2) The Neural Networks (NN)

Note: For more details on Neural Networks the reader can start to read the page 1165.

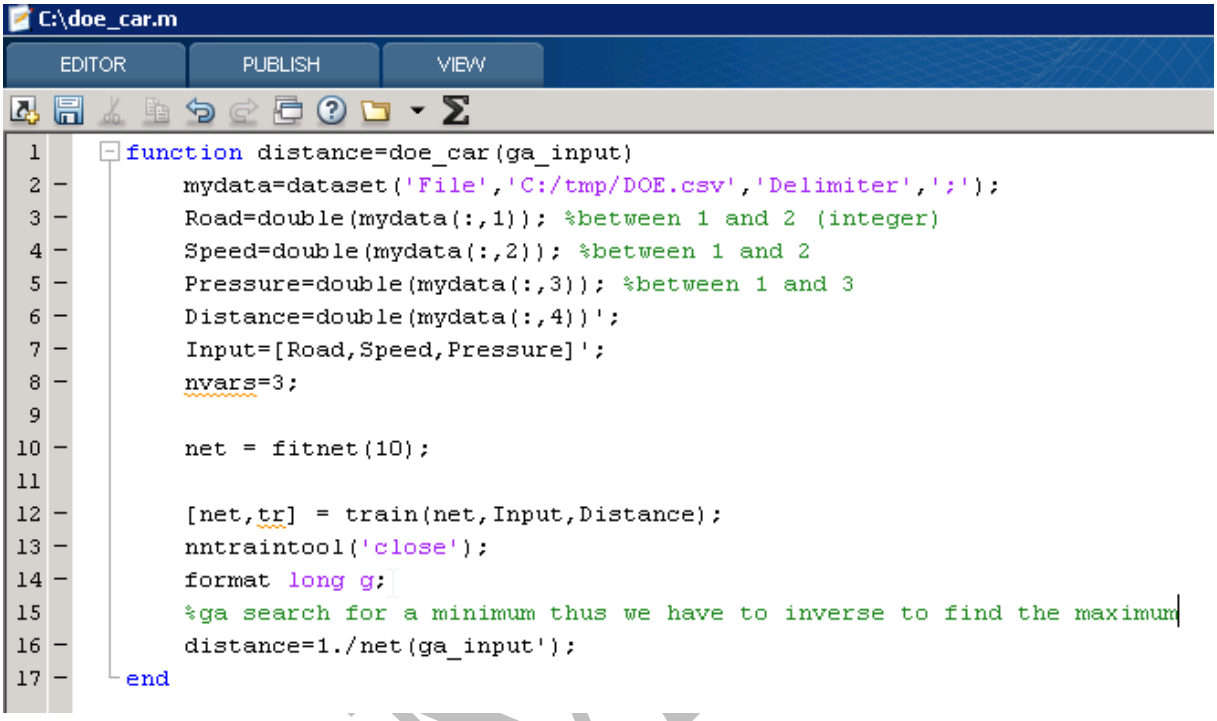
The purpose of this example is to analyze the famous full factorial dataset we used in the Design of Experiment theoretical course using the polynomial model:



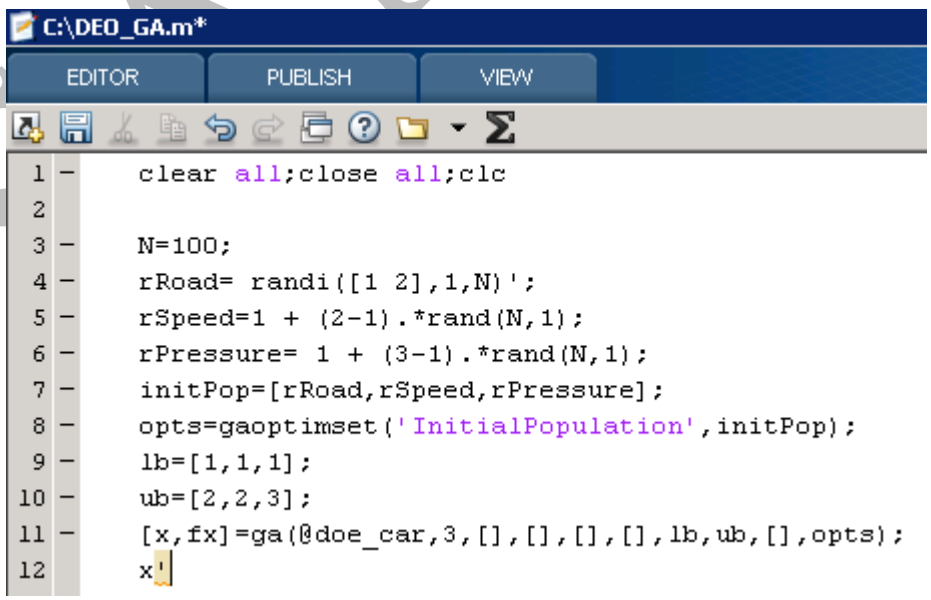
SAMPLE

And we want to find the optimum for this model using neural networks and GA as this is a famous method in high level R&D laboratories.

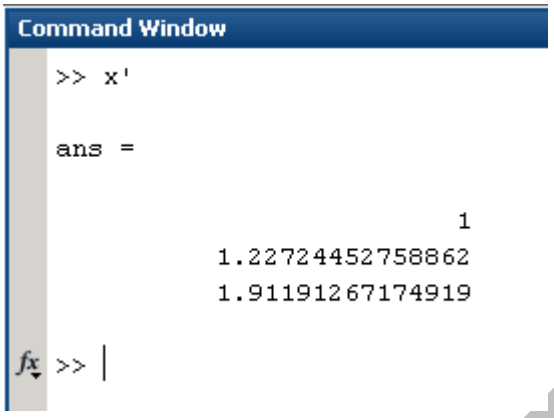
For this purpose we create a custom function deo_car( ) that will be called later by the **ga( )**. In this custom function doe_car( ) we use the functions **fitnet( )** with 10 hidden layers, after what we train the NN with the function **nntraintool( )**. We close neural network window tool with the command **nntraintool('close')**:



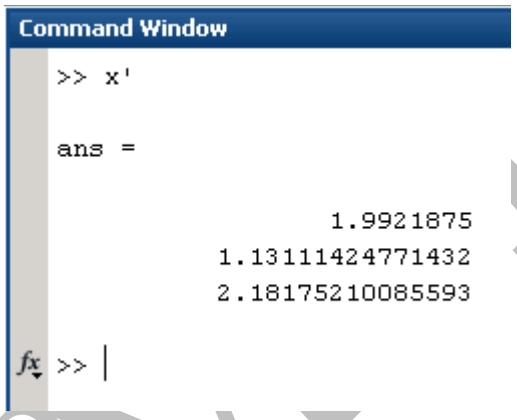
We generate a random population based on the range of values taken by the experiment for the type of Road, Speed and Pressure:



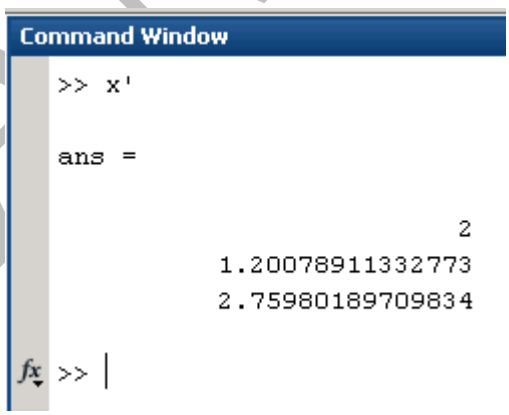
On the first run with N=100 we get:



On the second run with N=100, we get:

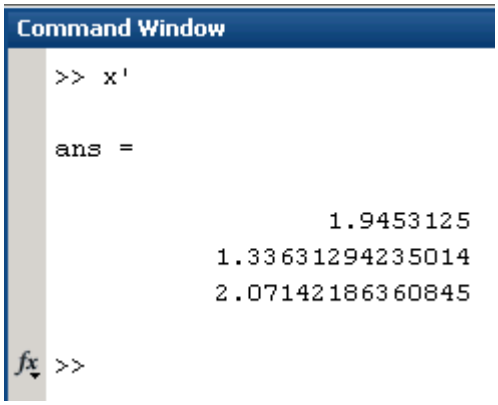


On the third run (we see that the answer fluctuates a lot with only 100 samples):

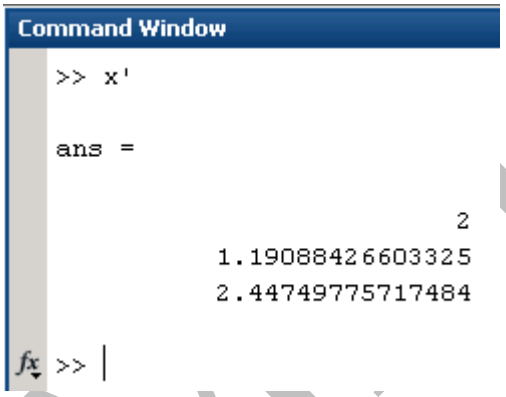


SAMPLE

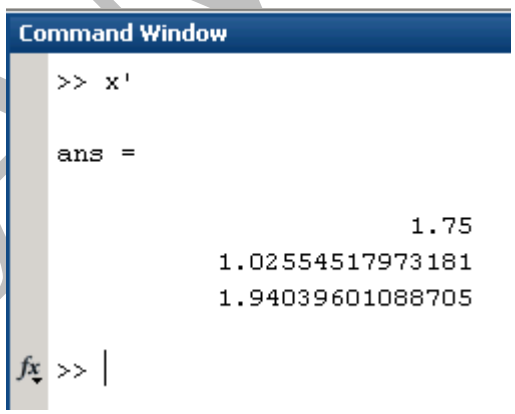
With 200 random samples:



With 300 random samples:

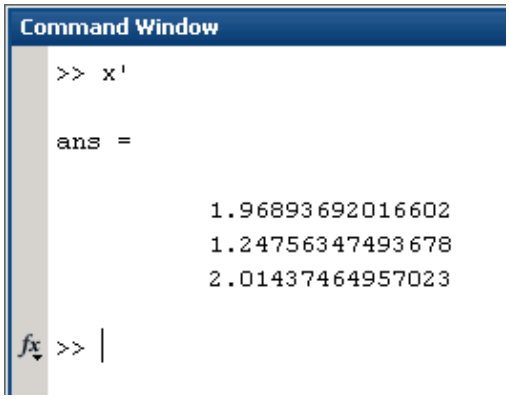


With 400 random samples:

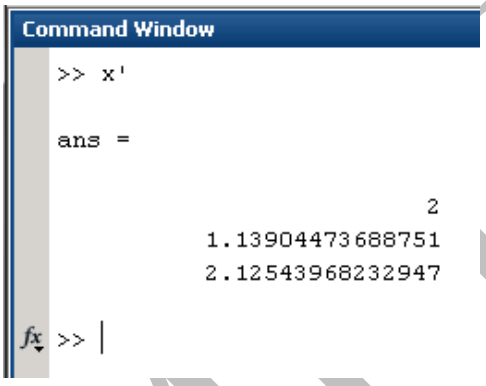


With 500 random samples:

SAMPLE



With 600 random samples:

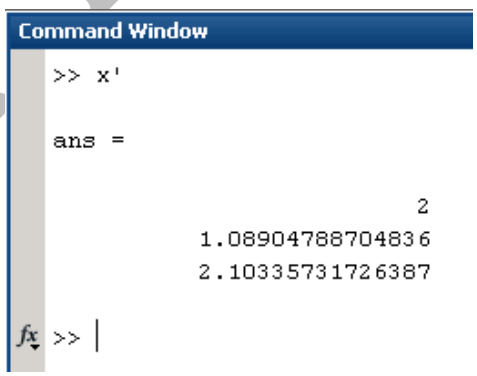


We see where the optimals inputs converge.

But we can do better and coming back to a sample of 100 but forcing the first variable (type of *Road*) to be an integer by a small change in the function **ga( )**:

```
[x,fx]=ga(@doe_car,3,[],[],[],[],lb,ub,[],[1],opts);
```

This gives us not a significant change:



Remember that in the theoretical DOE course we get using classical polynomial full factorial design the following values for the parameters:

[2 (road), 1 (50 [*km*/*h*]), 2 (2 [*kg*])]

and as we can see it, this corresponds almost perfectly!

SAMPLE

Finance (with Finance/Optimization and Econometrics Toolboxes)

Here we will just focus on the basics of the: Finance Toolbox, Statistics Toolbox, Optimization Toolbox, Trading Toolbox, Financial Instruments and Econometrics Toolboxes. That means mainly on topics typically covered during the CFAI/II/III or FRM studies.

For the new features of each version of the finance toolbox please refer to the following link:

http://www.MathWorks.ch/ch/help/finance/release-notes.html

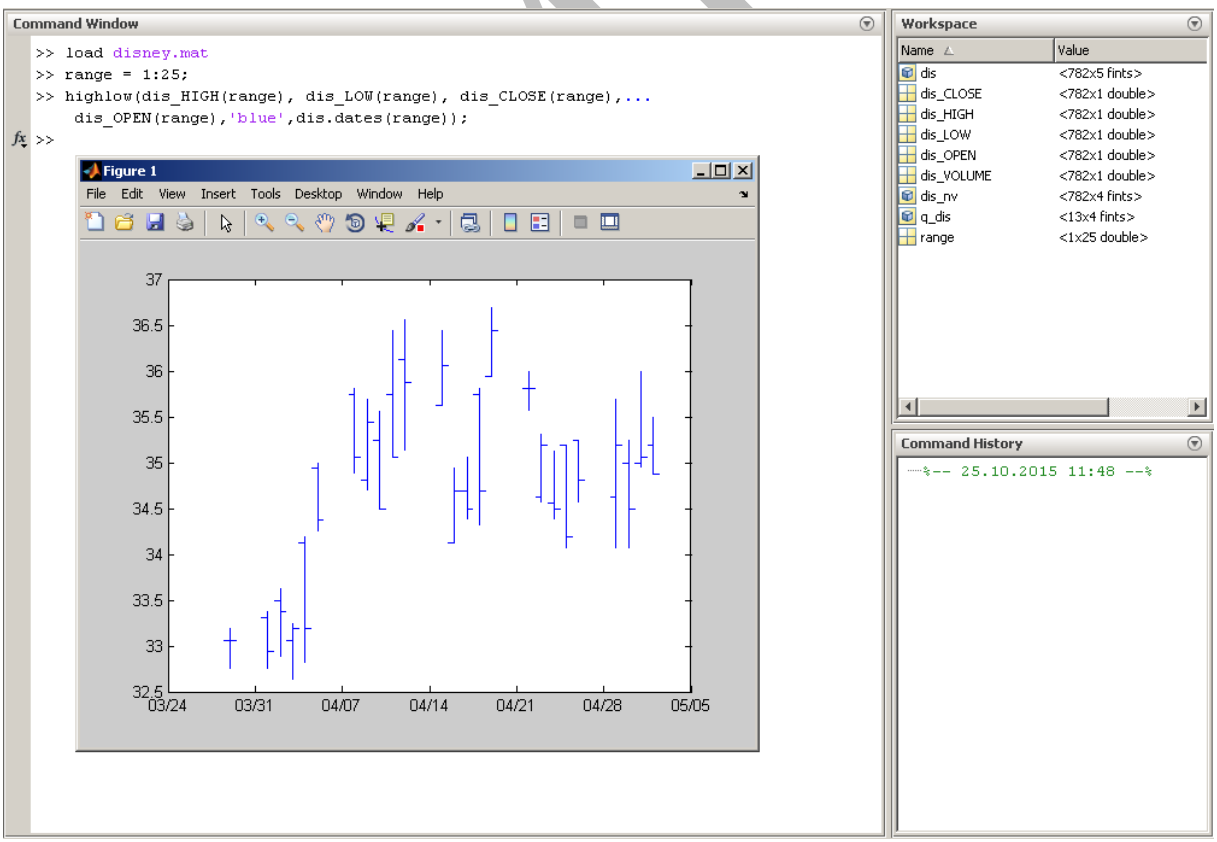
We will also study and practice in this chapter only stuff that we have study in the theoretical training on mathematical finance.

Plots

For sure you can with MATLAB plot easy traditional financial plots:

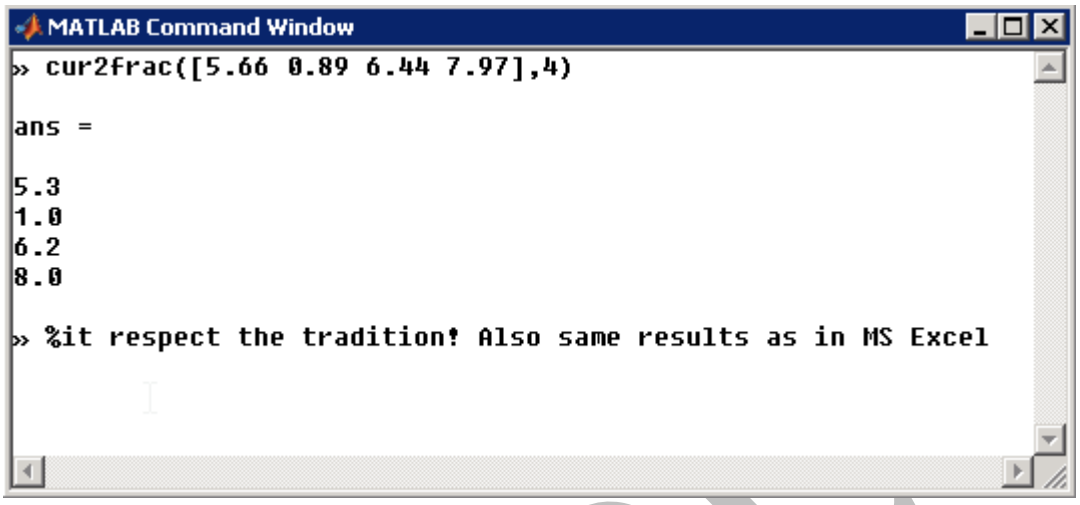
OHLC Chart (Open-High-Low-Close)

This is a very traditional chart in finance and don't need any explanations to be understand using the **highlow( )** function:



Fractional prices

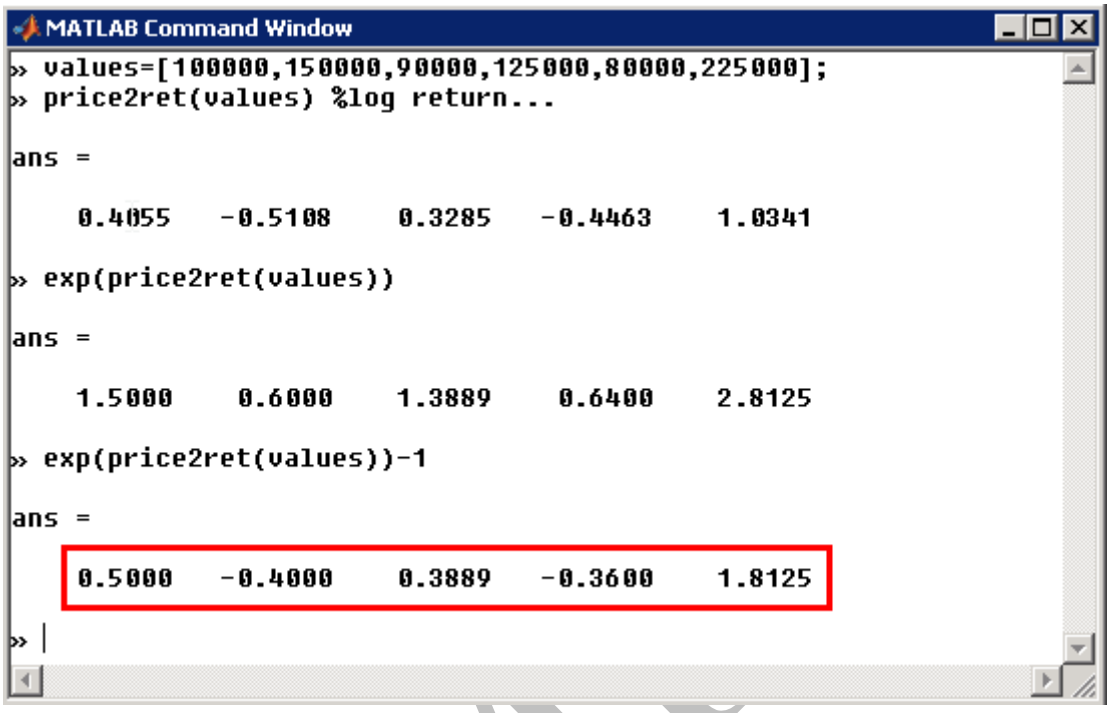
We begin by converting values to fractional one as we can see in newspapers using the function **cur2frac( )**:



SAMPL

Return

The purpose here is to convert values into returns and see what MathWorks calls "returns" using the function **price2ret( )**:

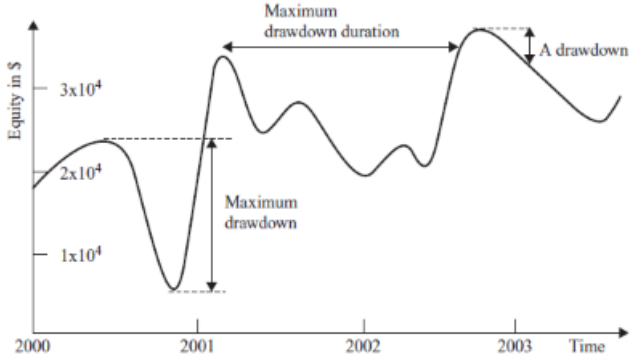


The last line rounded of red corresponds to what we get with the R software in the R course!

SAMPLE

Maximum Draw Down

No comment about this indicator...



Let us see how to calculate "by hand" what does the **maxdrawdown( )** function:

```
Command Window
>> values=[100000 150000 90000 125000 80000 225000];
>> MDD=maxdrawdown(values)

MDD =

    0.4667

>> 150000 %the value when the first drawdown will appear

ans =

      150000

>> 80000 %the last value befor a drawup appears

ans =

       80000

>> MDD=(150000-80000)/150000

MDD =

    0.4667
```

Or instead of having a ratio we can get the difference:

```
>> maxdrawdown(values,'arithmetic')

ans =

       70000

>>
```

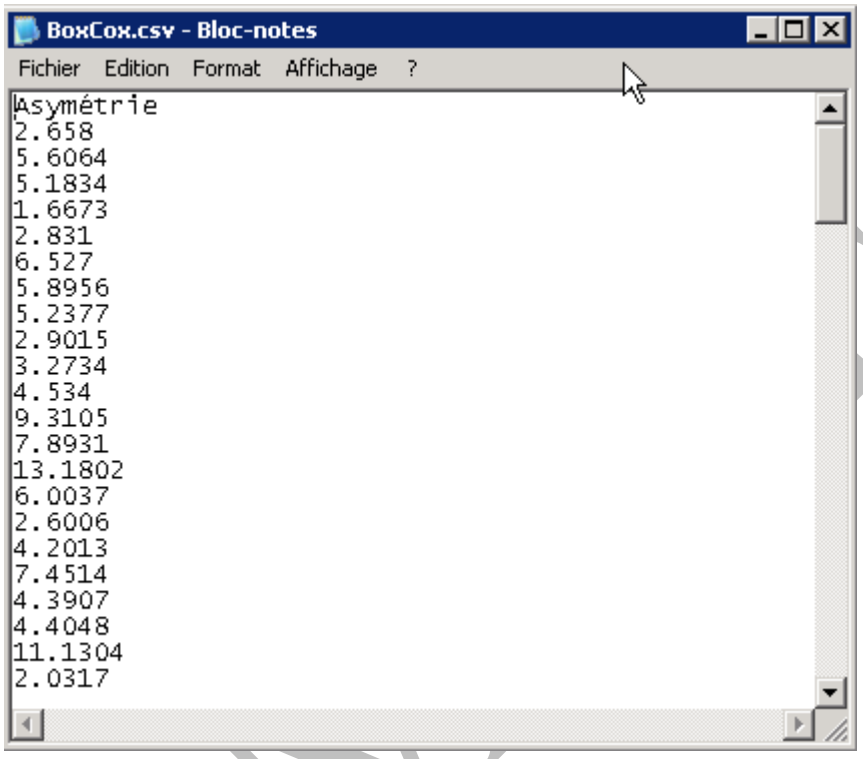
In both cases we obtain the same values as in the R software in the R course!

SAMPLE

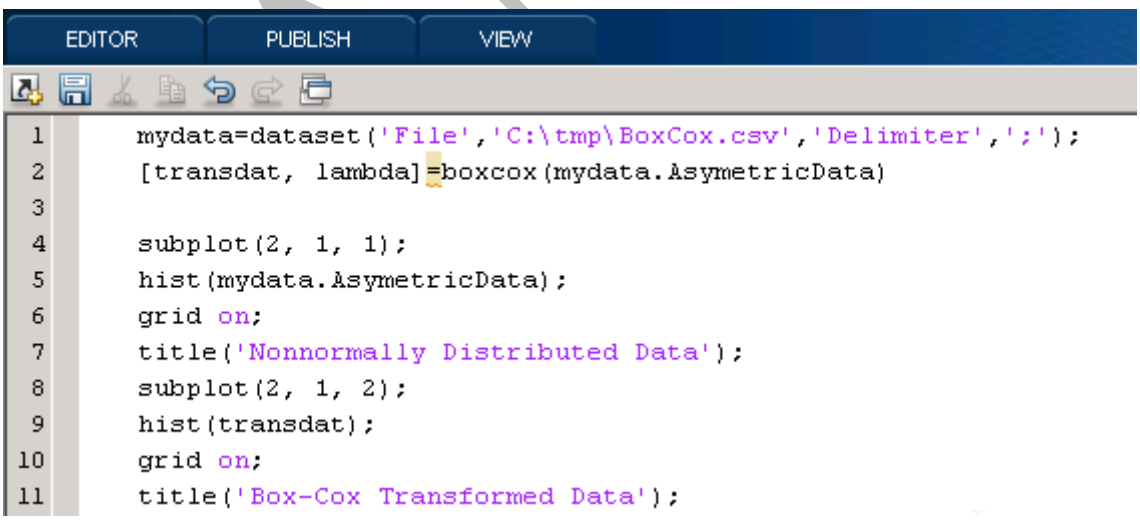
Box-Cox transformation

Caution!! Remember that if you have negative data Box-Cox transformation won't work. You will have to shift your data before and after shift the result back again.

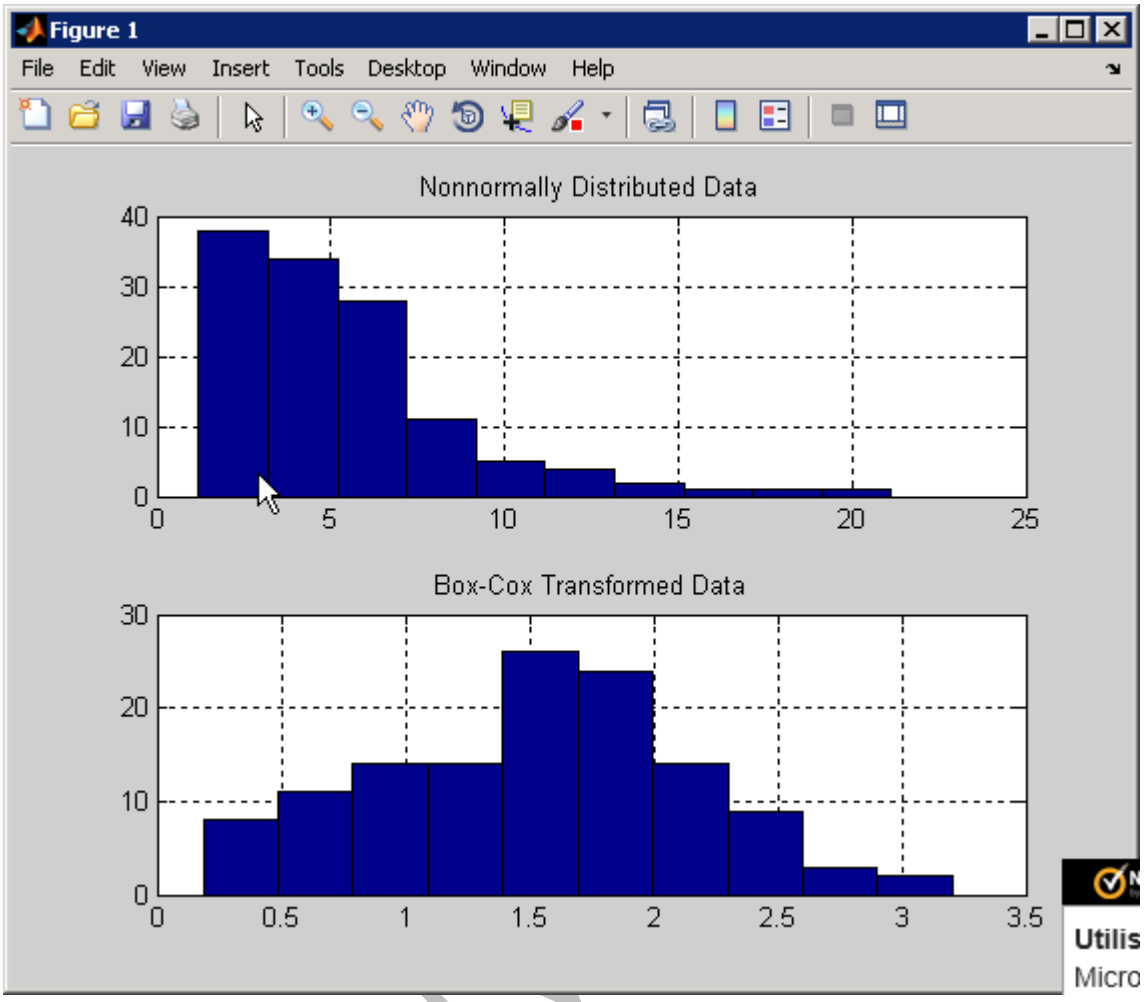
Consider the following asymetric (skewed) datas set:



We import the data and use the function **boxcox( )** the normalized the data:



We can see on the chart below that the result is quite good:



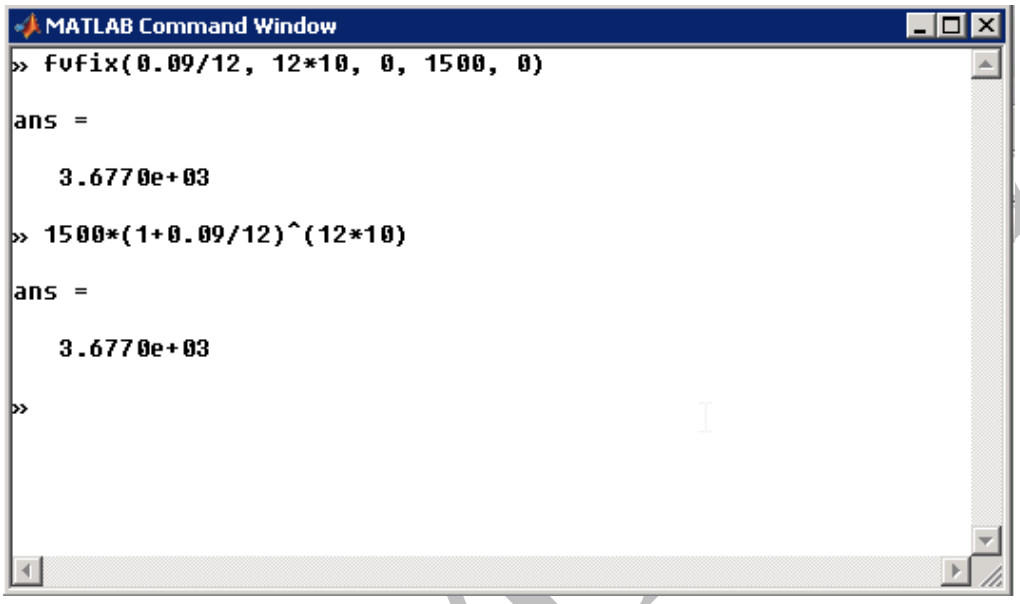
SAMPLE

Simple capital calculations

The calculations that follows now a very basic (high school level and even below). The purpose is just to calculate actual and future values of fixed rate investments.

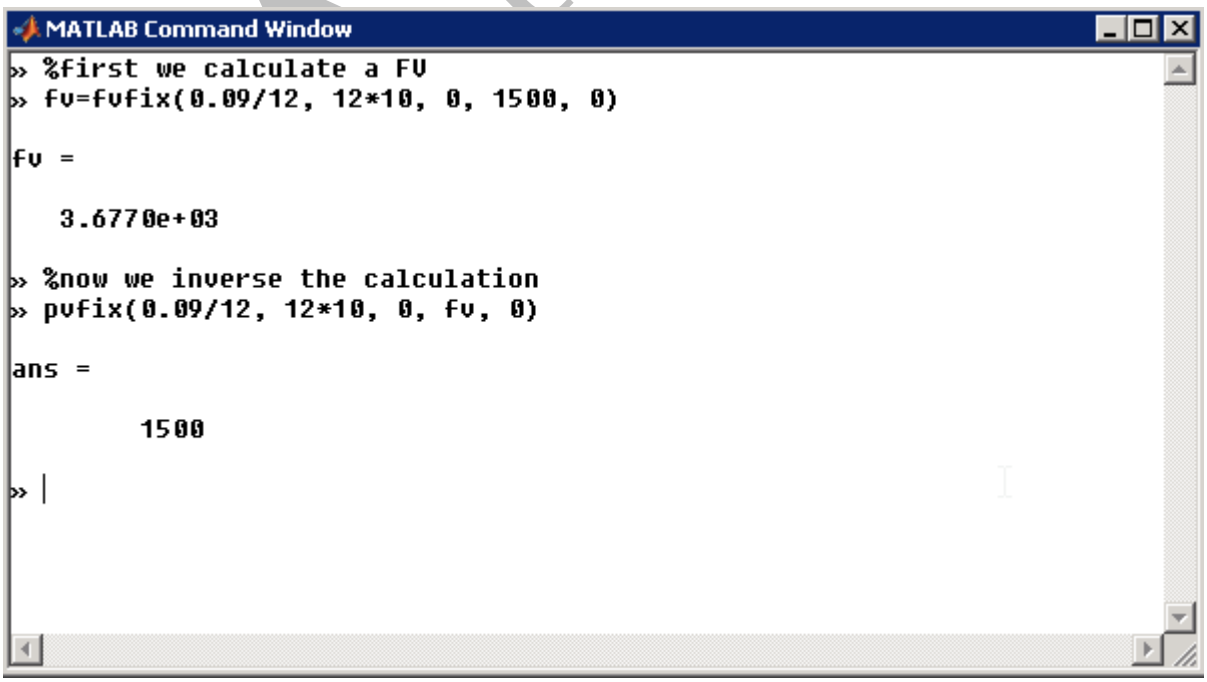
Composed Interest (fixed future value)

Here this is so simple that there is nothing to say about the usage of the function **fvfix( )**:



Composed Interest (fixed present value)

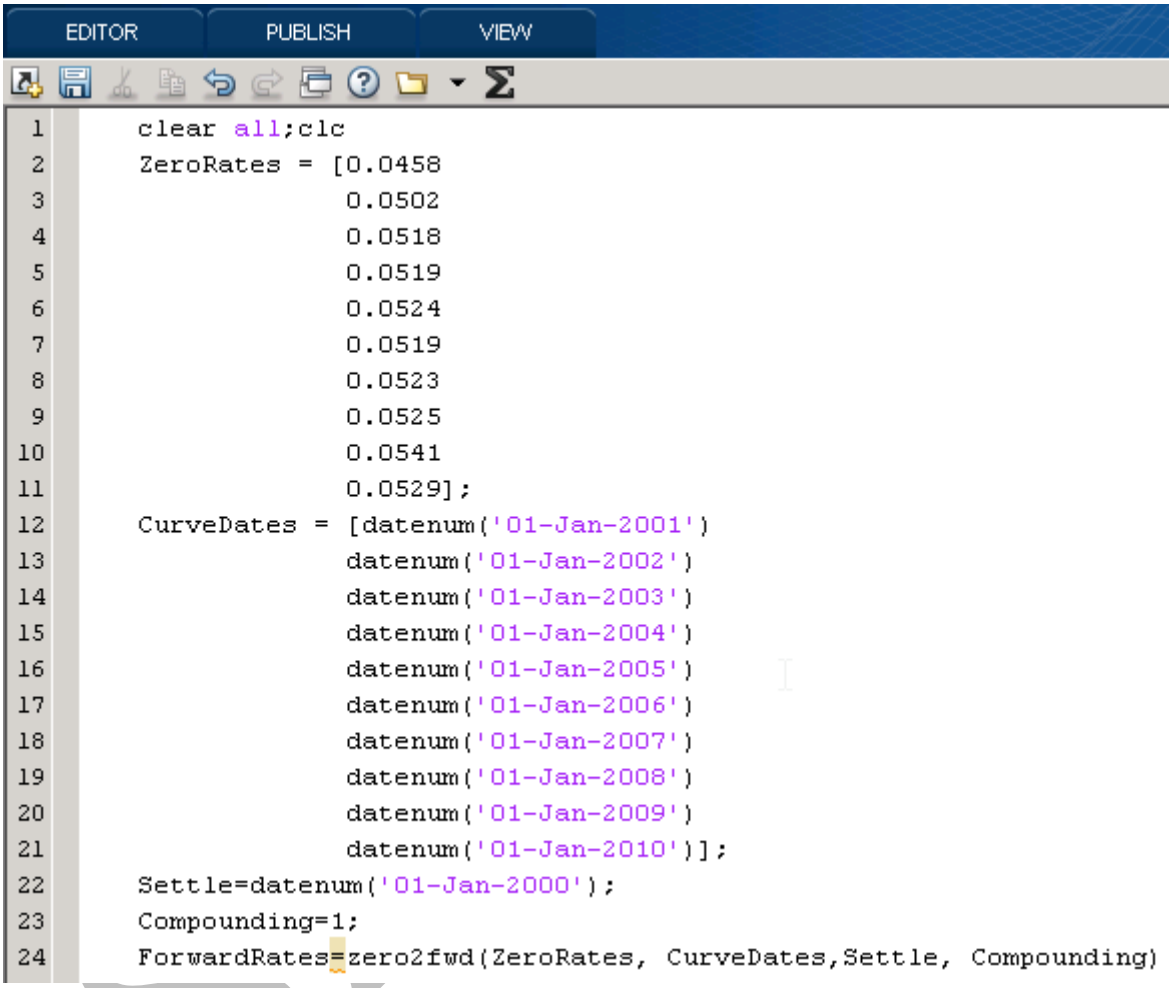
Here the purpose is just to invert the previous calculation using the function **pvfix( )**:



Spot rate (zero coupon rate) to forward rate

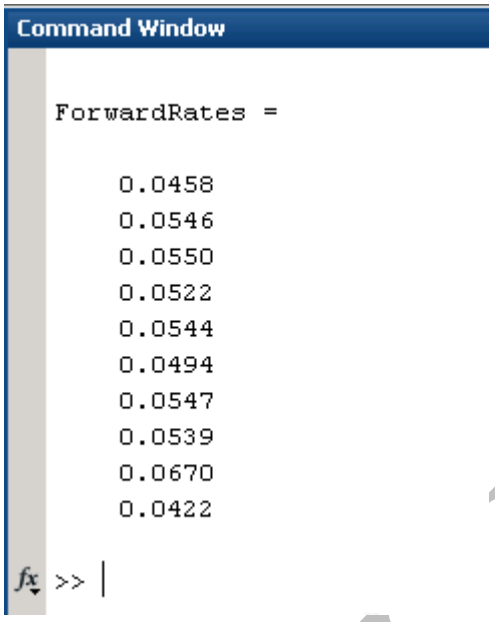
We have seen in the theoretical course that this is a major subject in modern portfolio theory. This topic is particularly difficult when the subject is to interpolate forward rate between known maturity dates (Nelson-Siegel and other interpolation models).

We will use the same data as those used in the theoretical course and apply the function **zero2fwd( )**:




```
clear all;clc
ZeroRates = [0.0458
             0.0502
             0.0518
             0.0519
             0.0524
             0.0519
             0.0523
             0.0525
             0.0541
             0.0529];
CurveDates = [datenum('01-Jan-2001')
              datenum('01-Jan-2002')
              datenum('01-Jan-2003')
              datenum('01-Jan-2004')
              datenum('01-Jan-2005')
              datenum('01-Jan-2006')
              datenum('01-Jan-2007')
              datenum('01-Jan-2008')
              datenum('01-Jan-2009')
              datenum('01-Jan-2010')];
Settle=datenum('01-Jan-2000');
Compounding=1;
ForwardRates=zero2fwd(ZeroRates, CurveDates,Settle, Compounding)
```

Thus we get



So we get exactly the same results as the calculation by hand!

For sure, we can invert the process using the function **fwd2zero( )**.

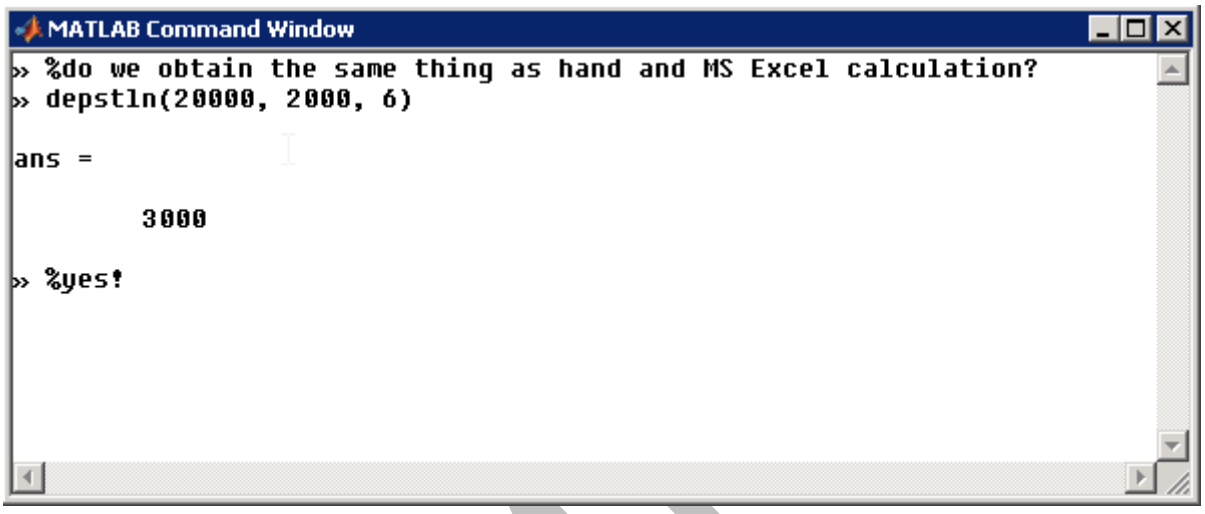
SAMPLE

Amortization and deprecation

The purpose here is just to reproduce the calculations made by hand and in Microsoft Excel after the proofs we have studied in the theoretical course.

Straight line deprecation (SLN)

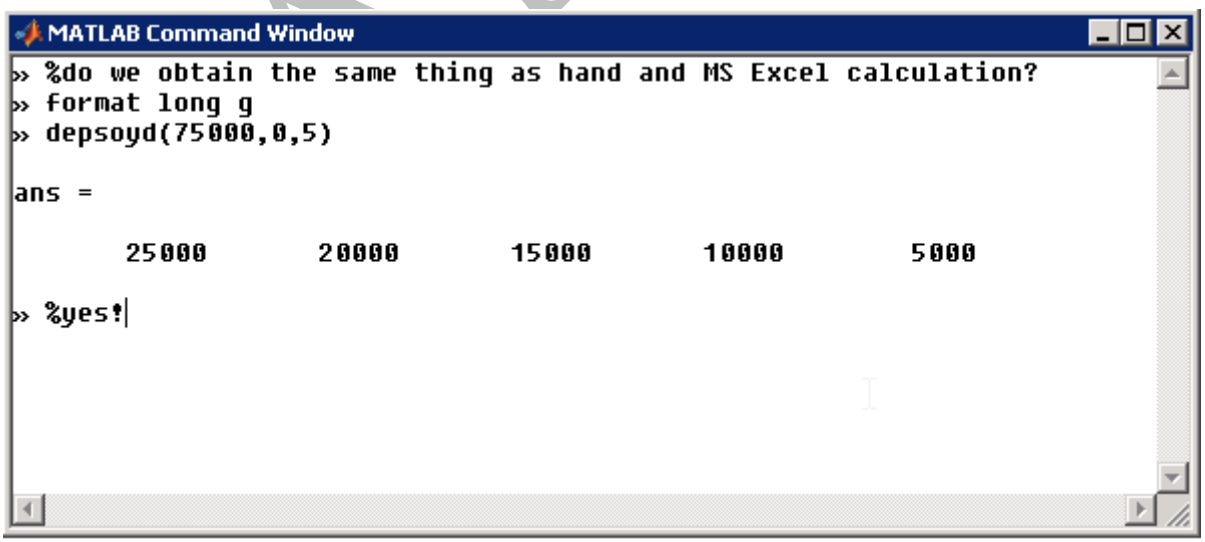
We begin with the first most intuitive function for depreciations **depstln( )**:



The results correspond perfectly to the calculations made by hand and with Microsoft Excel.

Sum of years digit (SYD)

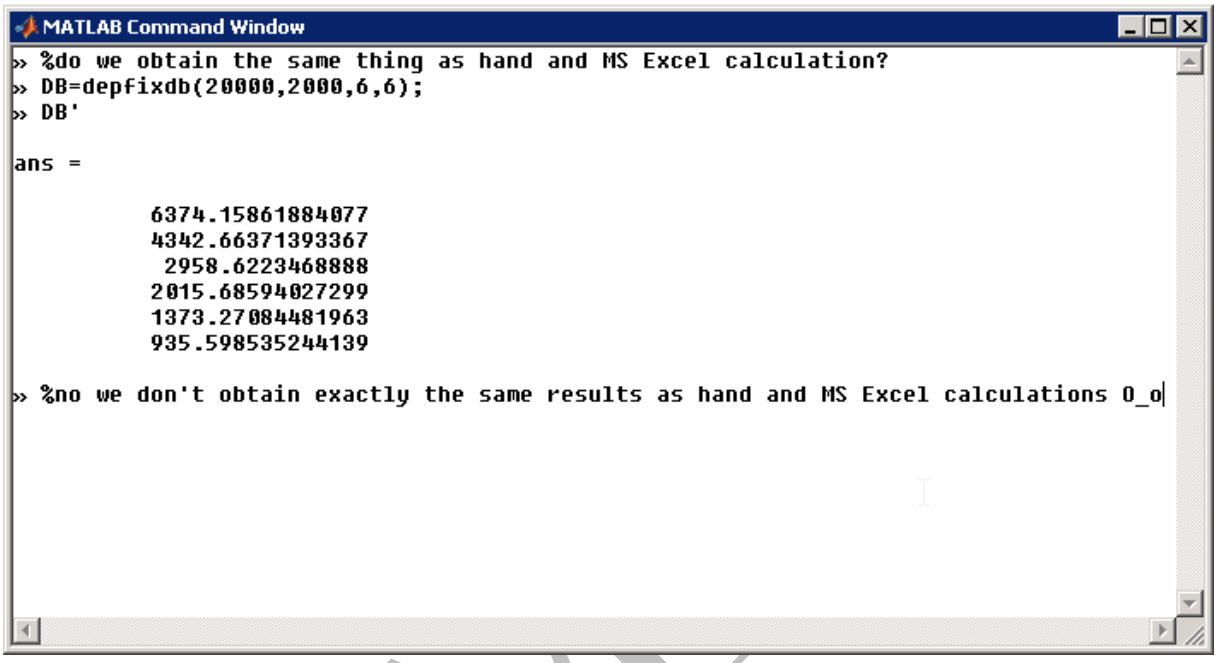
We know it is an empirical rule but we still check if we found the same results as the calculations made by hand and with Microsoft Excel using the function **depsoyd( )**:



Fixed declining-balance depreciation schedule (DB)

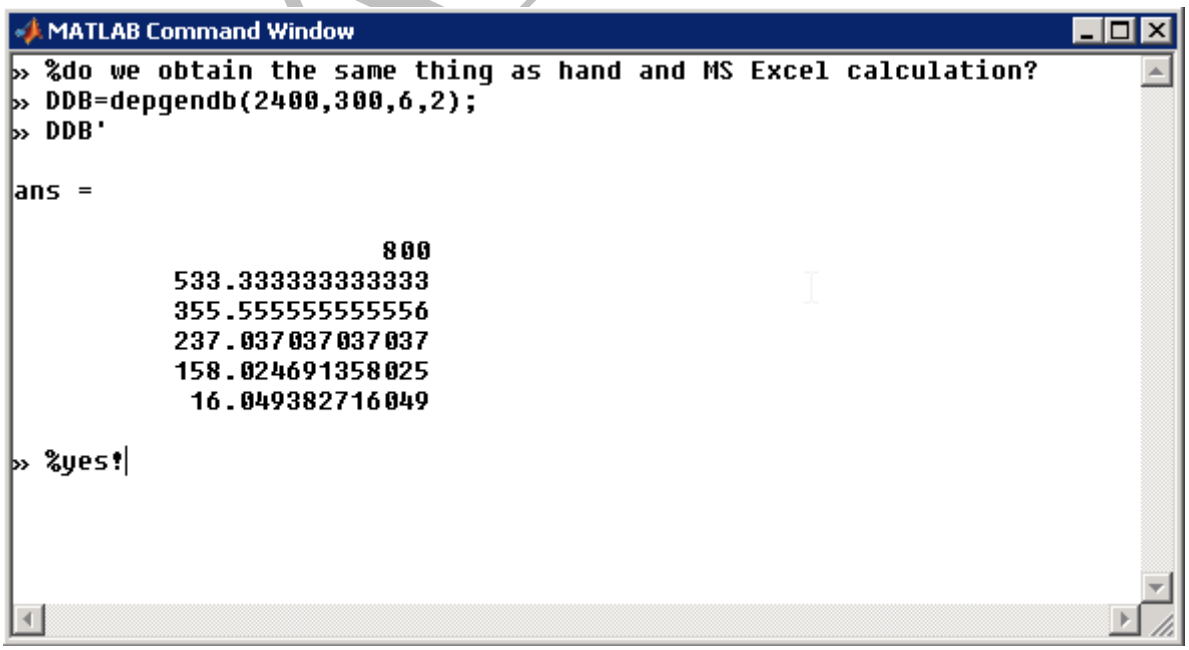
We have seen and proved in the theoretical course that the DB depreciation is an intuitive method (at the opposite of SYD that is an empirical U.S. rule).

Let us have a look if we have the same results as by hand and Microsoft Excel using the function **depfixdb( )**:



Double declining balance (DDB)

We have seen and proved in the theoretical course that the DDB depreciation is also an intuitive method. Let us check that with the function **depgendb( )**:

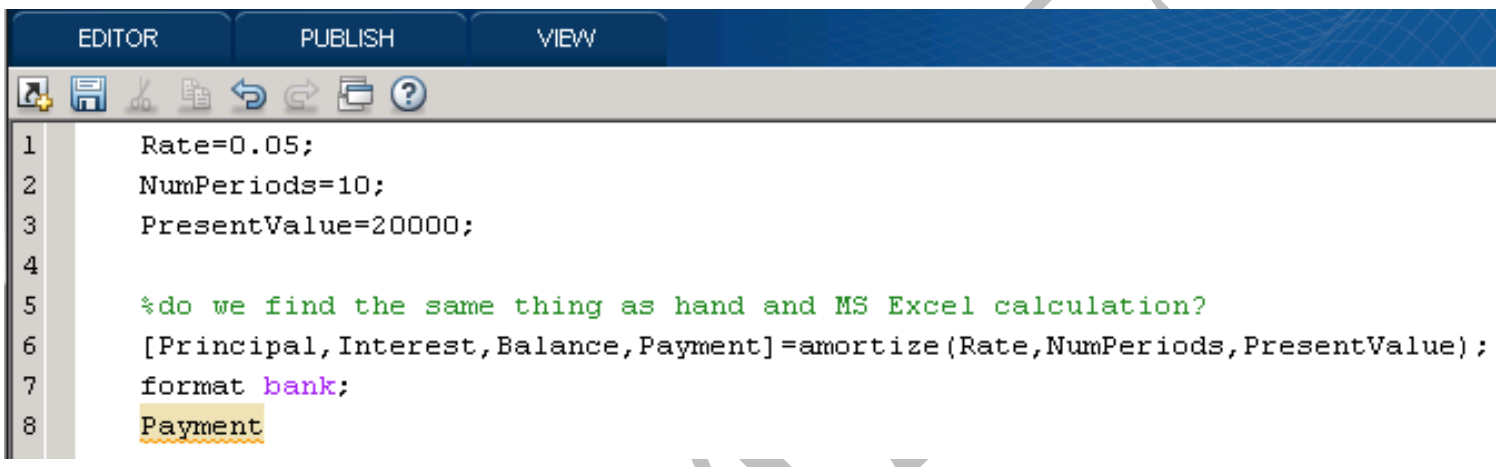


Loans and credits

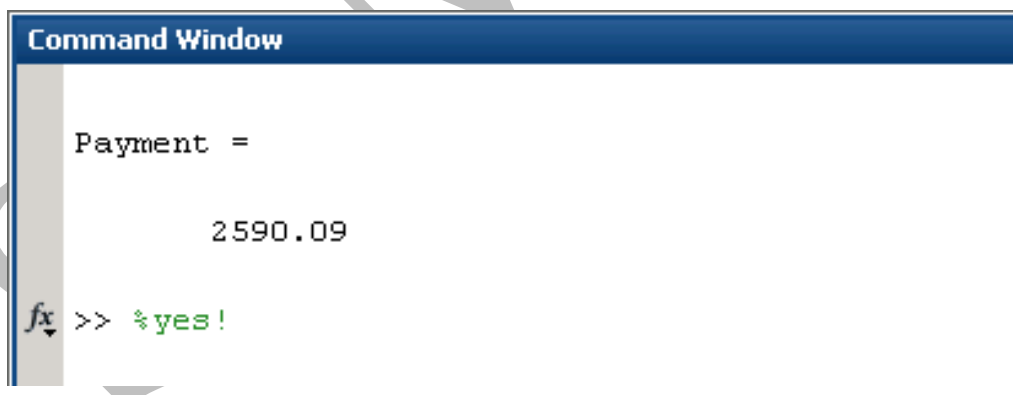
All calculations here are high school mathematics level and have been proven in the theoretical Accounting course and applied by hand and also with Microsoft Excel. The purpose here will be as always to be sure that we found the same results for each example.

Payment for a loan based on constant payments and a constant interest rate (PMT)

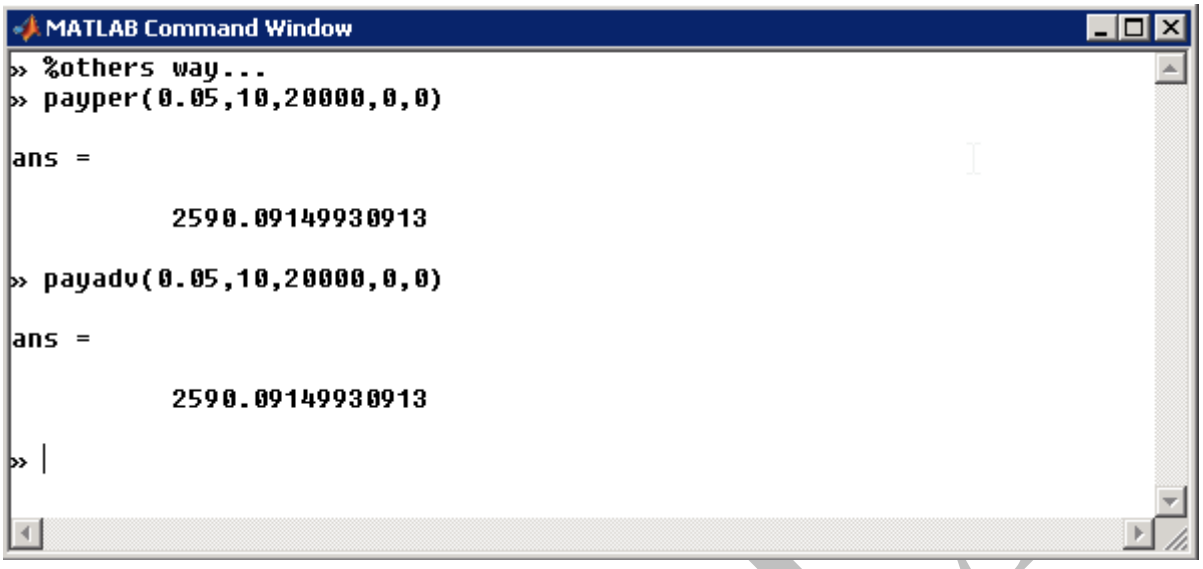
We begin with the famous PMT (as named in Microsoft Excel) where the function in MATLAB is **amortize( )**:



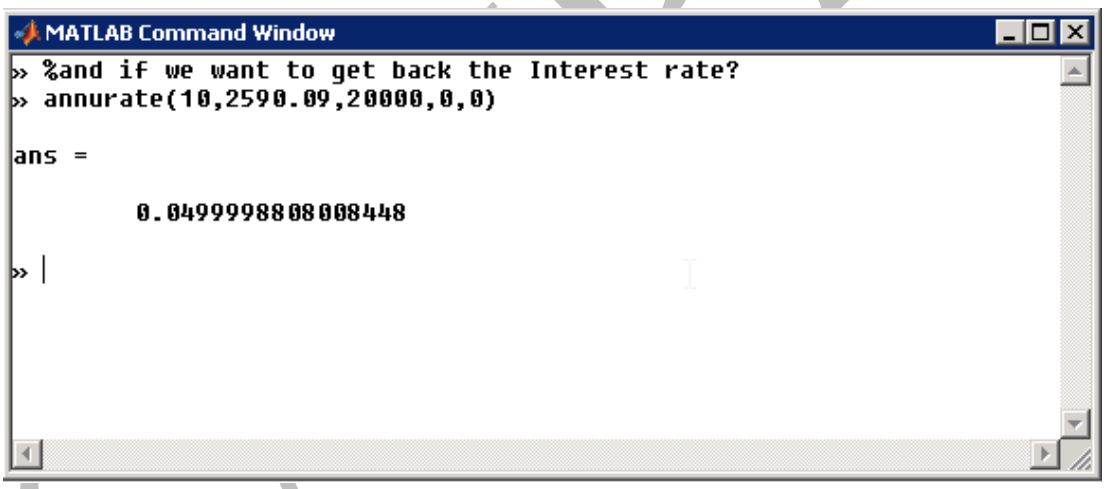
As we can see it we obtain the same result as the calculation made my hand and with Microsoft Excel:



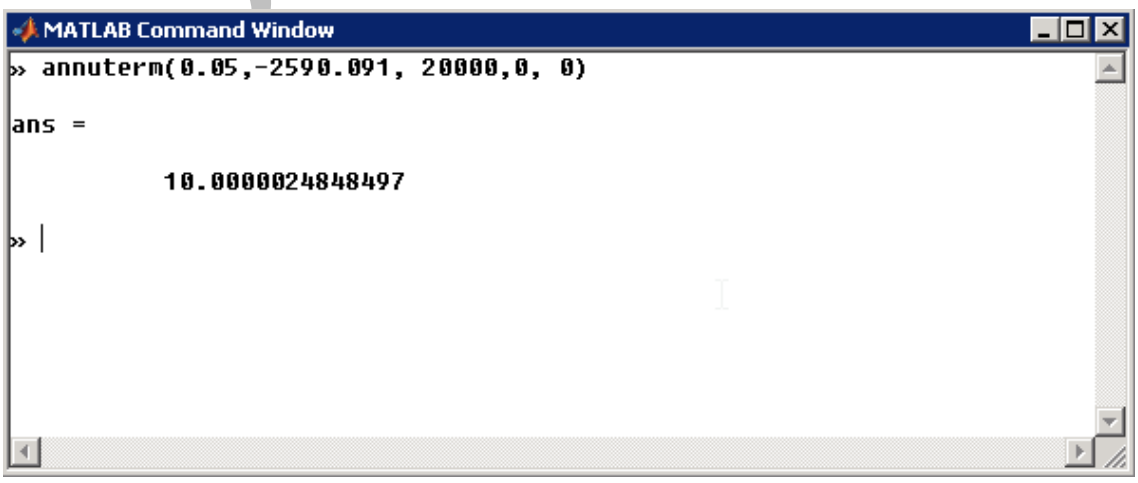
As in Microsoft Excel we can use different function to get the same results using the functions that **payper( )** or **payadv( )**:



And also like in Microsoft Excel and by hand we can invert the problem by asking what is the interest rate when we know all others parameters using the function **annurate( )**:

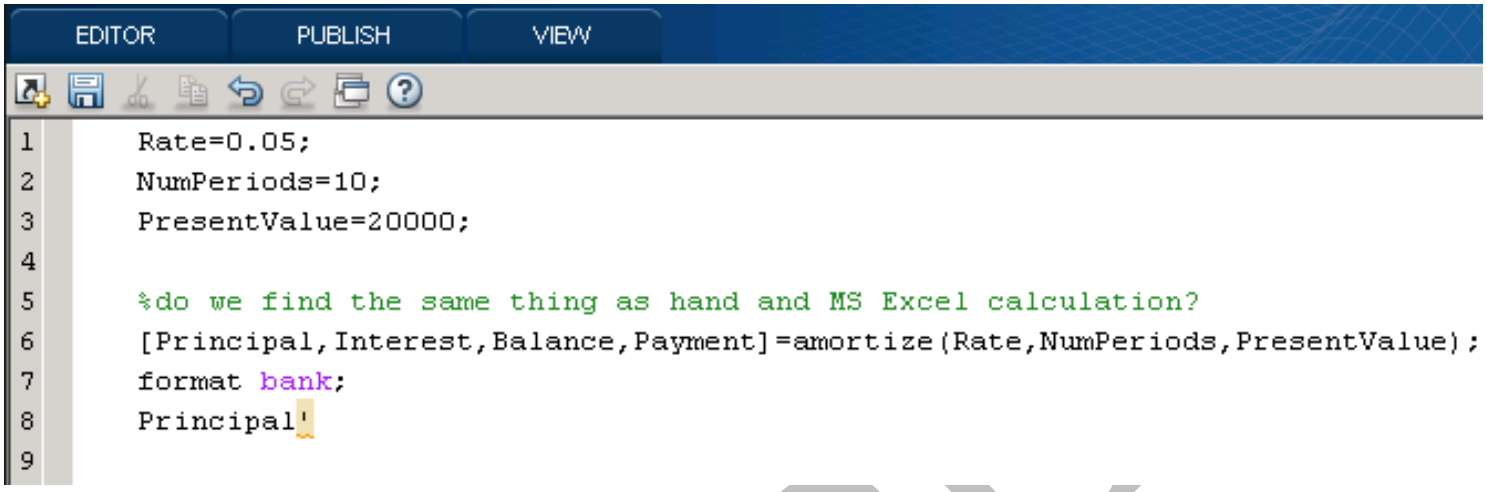


Or also the number of periods using the function **annuterm( )**:

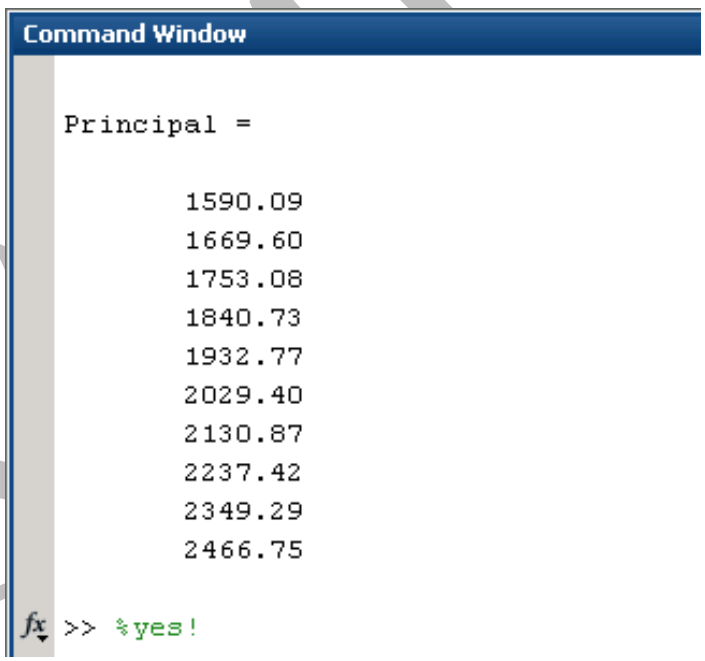


Principal Payment (PPMT)

We stay with the function **amortize( )** to see now the different principal values for each month and see if the correspond to the calculations made by hand and with Microsoft Excel (using PPMT( ) function):



As we can see it we obtain the same result as the calculations made my hand and with Microsoft Excel:



Interest Payment (IPMT)

We stay with the function **amortize( )** to see now the different interest we pay for each month and see if the correspond to the calculations made by hand and with Microsoft Excel:

EDITOR PUBLISH VIEW

```
1  Rate=0.05;
2  NumPeriods=10;
3  PresentValue=20000;
4
5  %do we find the same thing as hand and MS Excel calculation?
6  [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);
7  format bank;
8  Interest=Interest'
```

As we can see it we obtain the same result as the calculations made my hand and with Microsoft Excel:

Command Window

```
Interest =

       1000.00
        920.50
        837.02
        749.36
        657.33
        560.69
        459.22
        352.67
        240.80
        123.34

>> %yes!
```

Payment Present Value (PV)

We check here by using the function **amortize( )** that we get the same residual value of a credit. The value that we get here must simple be equivalent to the 20'000 where we substract the value we get before with the Principal Payement Value (see page 999) and that we can get directly with Microsoft Excel using the function **PV( )**:

EDITOR PUBLISH VIEW

```
1  Rate=0.05;
2  NumPeriods=10;
3  PresentValue=20000;
4
5  %do we find the same thing as hand and MS Excel calculation?
6  [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);
7  format bank;
8  PresentValue=Balance'
```

So we get:

Command Window

```
PresentValue =

      18409.91
      16740.31
      14987.24
      13146.51
      11213.74
       9184.34
       7053.46
       4816.04
       2466.75
          0.00

fx >> %yes!
```

And as we know, this is 100% correct.

Plot resume of loan payment

This can be seen just as an exercise for fun that resume plotting skills and one of the previous function we have seen just above:

EDITOR PUBLISH VIEW

```
Rate=0.05;
NumPeriods=10;
PresentValue=20000;

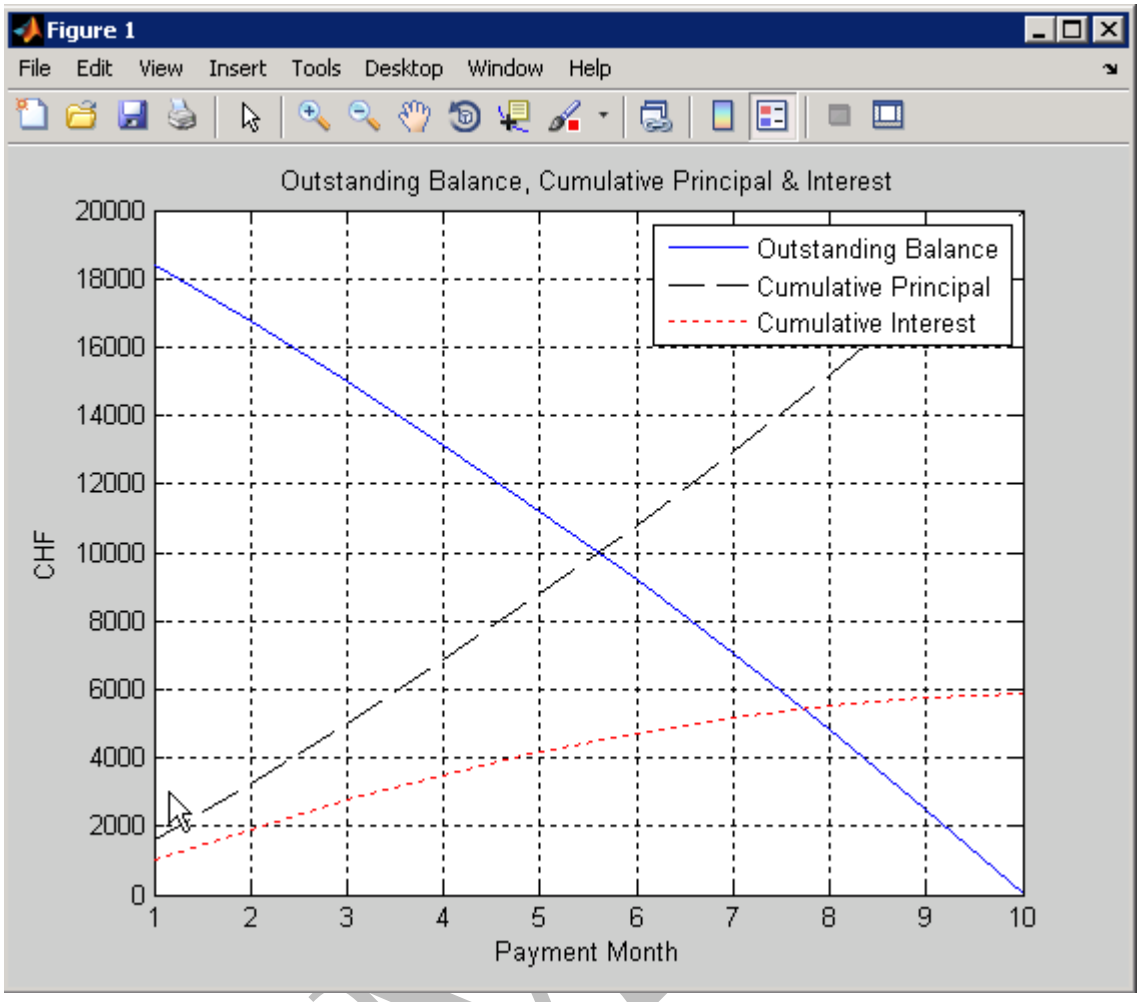
%do we find the same thing as hand and MS Excel calculation?
[Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);

plot(Balance,'b'), hold('on')
plot(cumsum(Principal),'--k')
plot(cumsum(Interest),':r')

xlabel('Payment Month');ylabel('CHF')
grid('on');
title('Outstanding Balance, Cumulative Principal & Interest')
legend('Outstanding Balance','Cumulative Principal','Cumulative Interest')
y=0:2000:20000; % define the y values where you want to have a tick
set(gca,'YTick',y);  % Apply the ticks to the current axes
set(gca,'YTickLabel',arrayfun(@(v)sprintf('%d',v),y,'UniformOutput',false));
```

Se we get the famous chart:

SAM

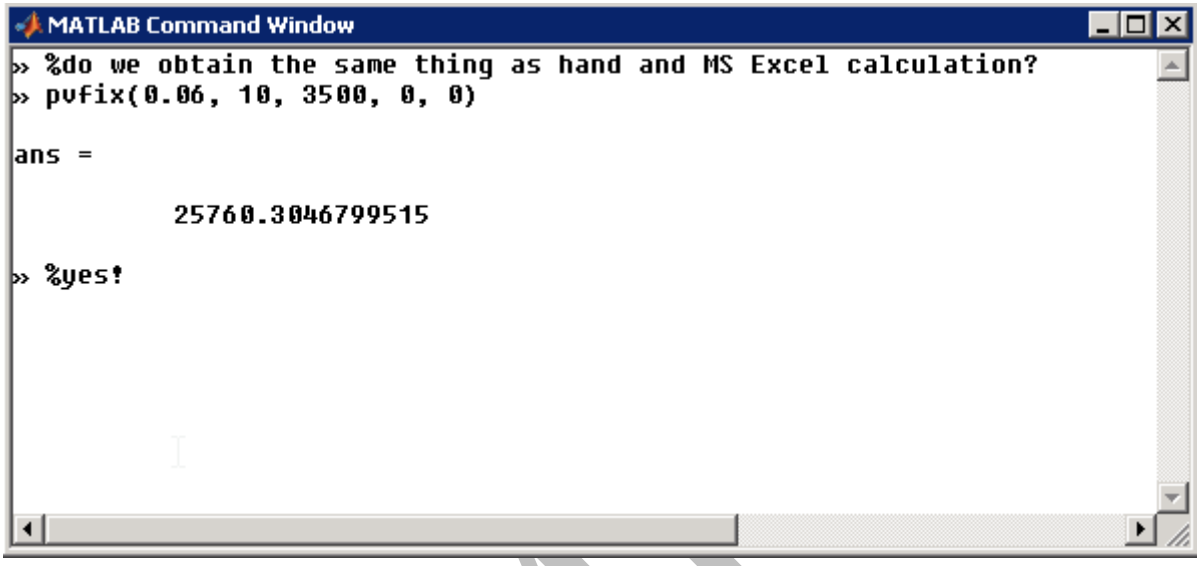

Figure 1
File Edit View Insert Tools Desktop Window Help
Outstanding Balance, Cumulative Principal & Interest
Outstanding Balance
Cumulative Principal
Cumulative Interest
CHF
Payment Month
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
1
2
3
4
5
6
7
8
9
10

SAM

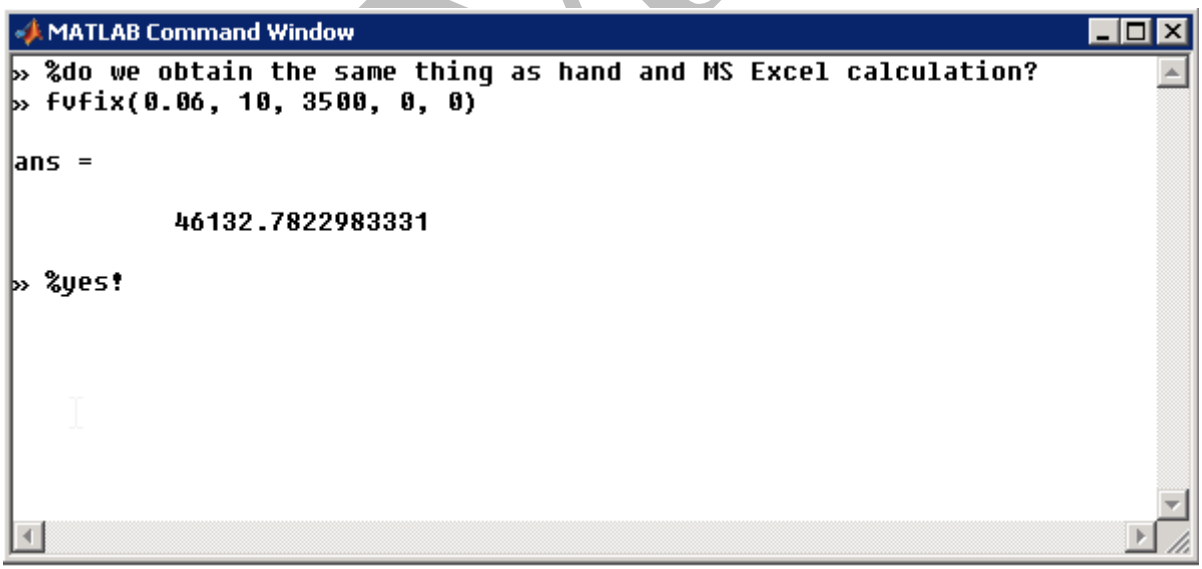
Annuities

As we know actuarial calculations are very important in insurance (they are the basics). Once again here we will check that we get the same results as those obtained by hand and with Microsoft Excel.

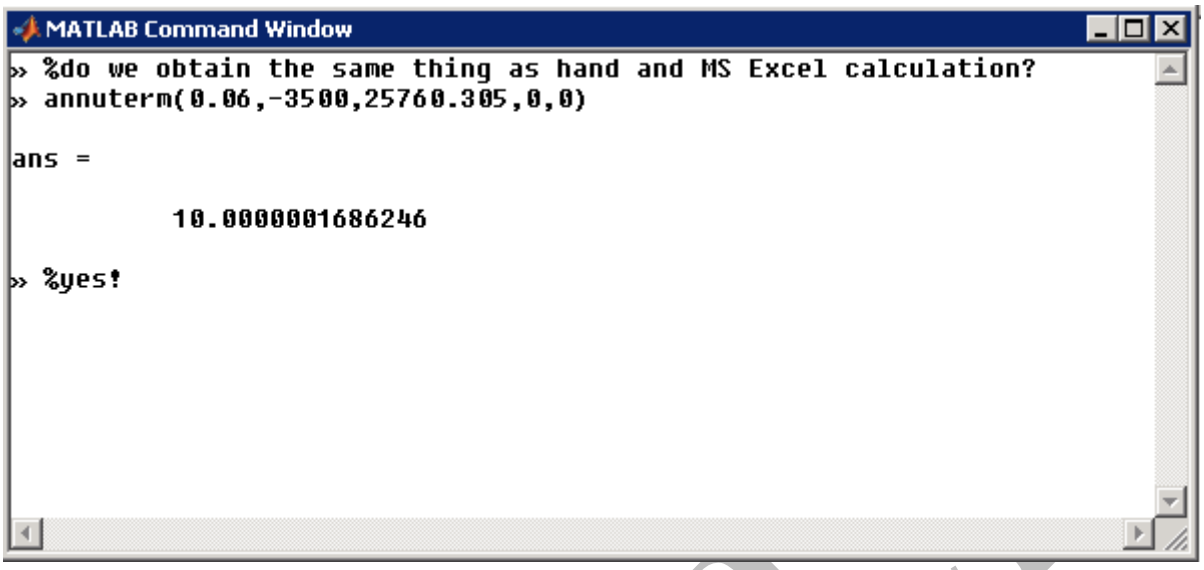
Postnumerando actual annuity value (PV)



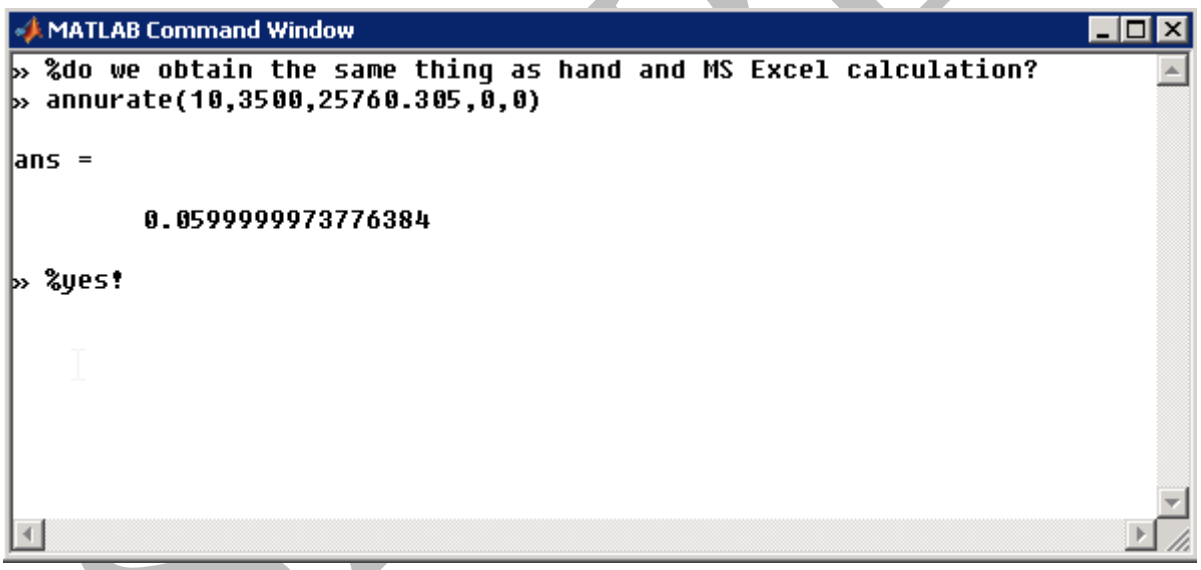
Postnumerando forward annuity value (PV)



Postnumerando number of periods (NPER)



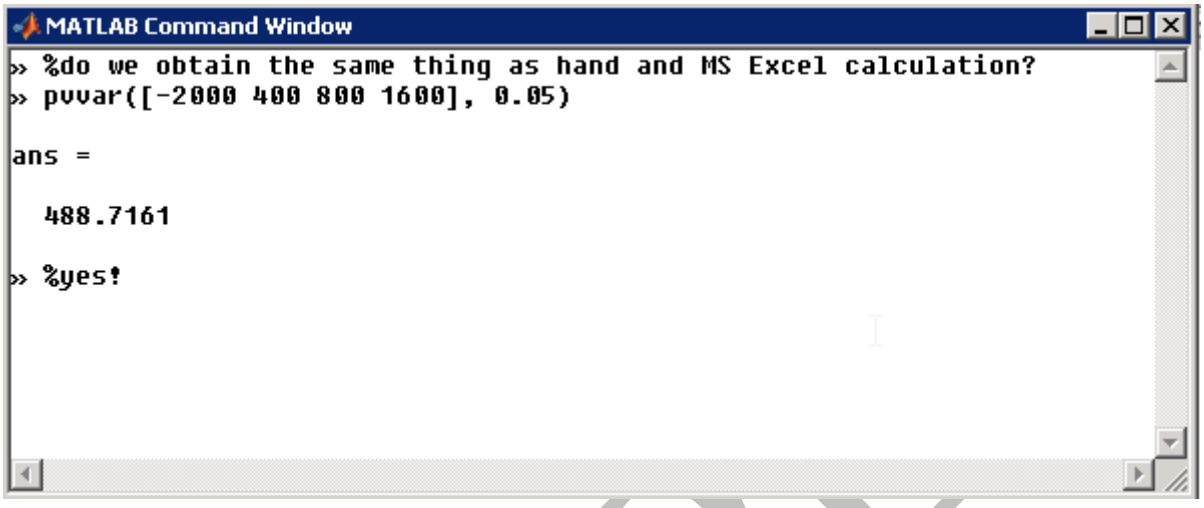
Postnumerando annuity rate (ANNURATE)



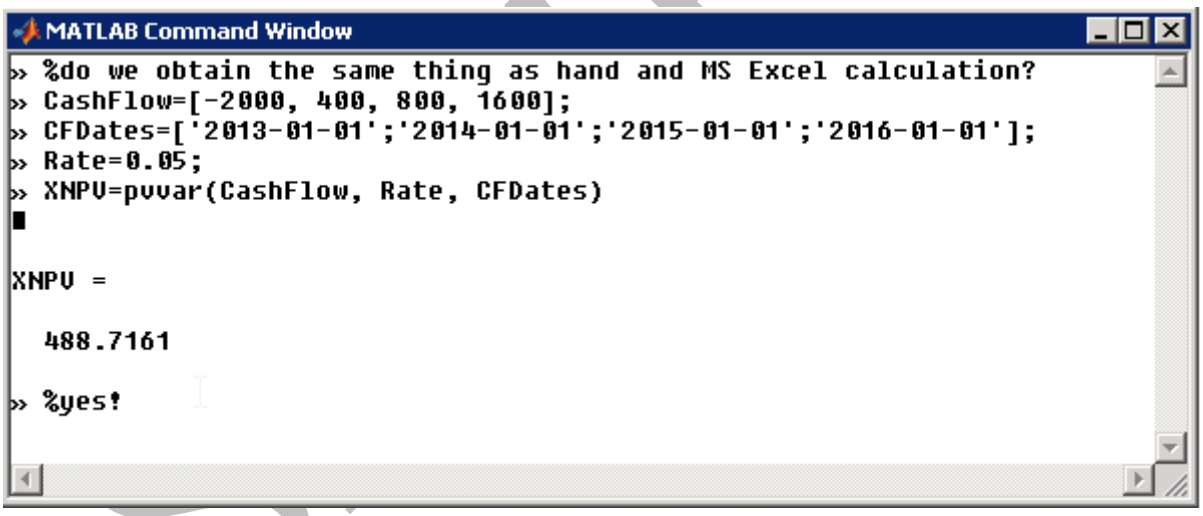
Corporate finance

Net Present Value (NPV)

Periodic NPV

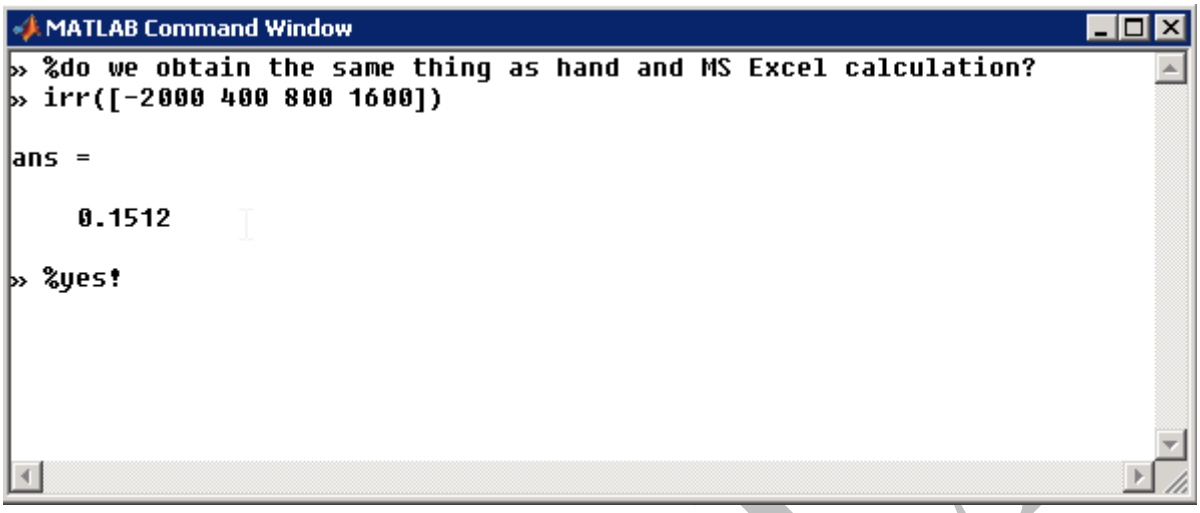


non-Periodic NPV

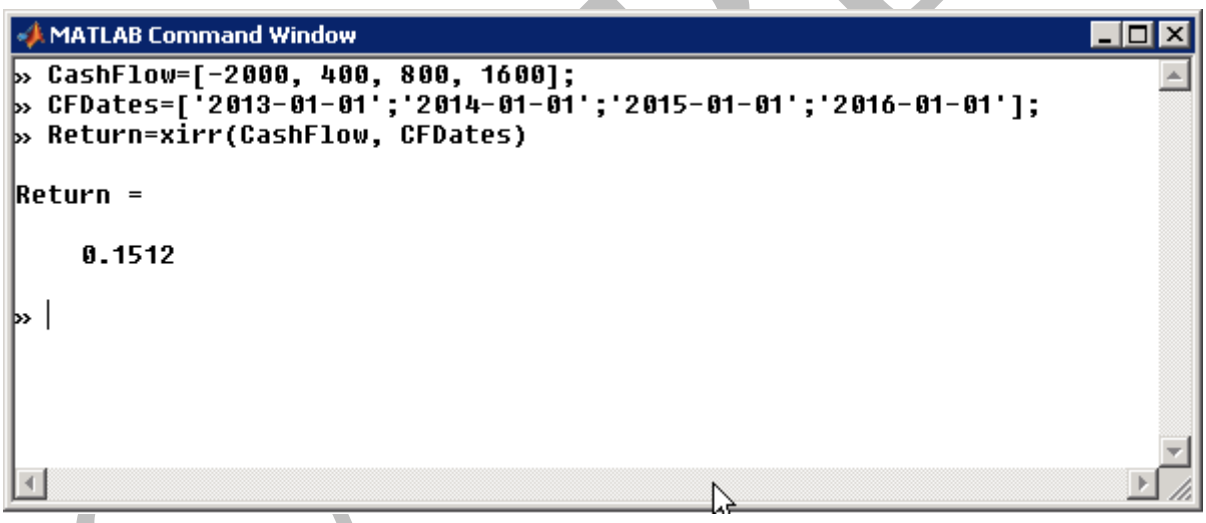


Internal Rate of Return (IRR)

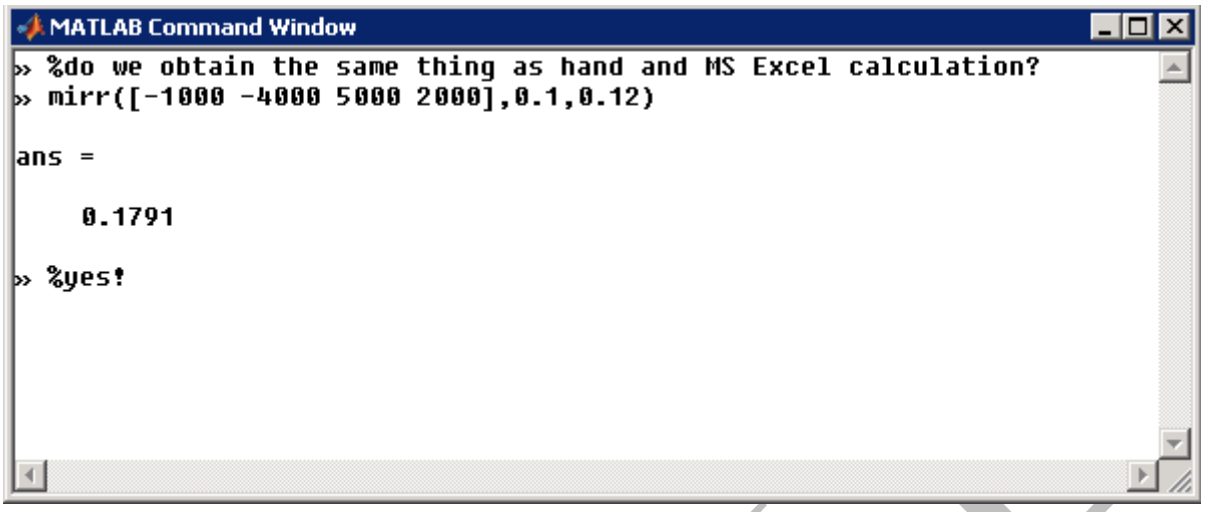
Periodic IRR



non-Periodic IRR



Modified Internal Rate of Return (MIRR)



SAMPLE

Treasury bills/Securities (bonds) calculations

Clean price of a security (BNDPRICE)

EDITOR PUBLISH VIEW

```
% This bndprice function returns Clean prices and accured interest due.
clear all;clc
%we will see if we found the same result as in the MS Excel training
Yld = 0.1; %Yield
CouponRate=0.06; % Coupon Rate
StartDate='2013-01-01';%Settlement date
EndDate='2017-01-01';% Maturity date
Period=12; %frequency
%note that a Basis of "1" in Excel corresponds to a Basis of "0" in Matlab
CleanPrice=bndprice(Yld,CouponRate,StartDate,EndDate,Period,'Basis',0)

%the result will be different from Microsoft Excel and calculations made by hand!!!
%I have to found why
```

Command Window

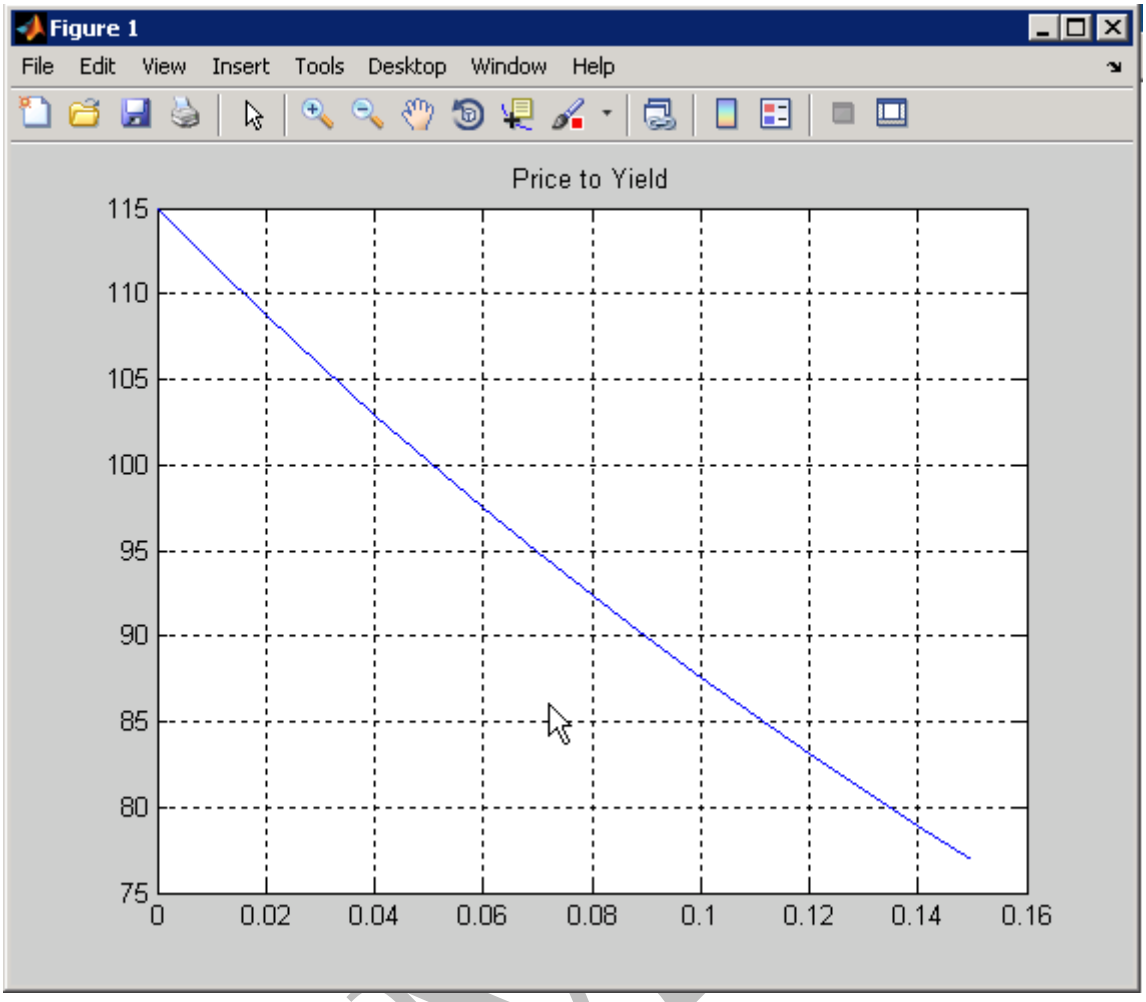
```
CleanPrice =

   87.4767

>>
```

EDITOR PUBLISH VIEW

```
%array of yields
yields=0:.01:.15;
% coupon rate
couponRates=.05;
% settlment date specified as today's date
sttl=datenum(today);
%specification of maturity date
maturity=datenum('1/01/2018');
%bndprice produces clean price and accrued interest with face value default
%of 100
[Price,Accruedint]=bndprice(yields,couponRates,sttl,maturity);
plot(yields,Price+Accruedint);
title('Price to Yield');
grid on;
```



Price of treasury bond/security (TBILLPRICE)

```
MATLAB Command Window
» %do we obtain the same thing as hand and MS Excel calculation?
» tbillprice(0.09,'1999-03-30','1999-06-01',2)

ans =

           98.470337496965

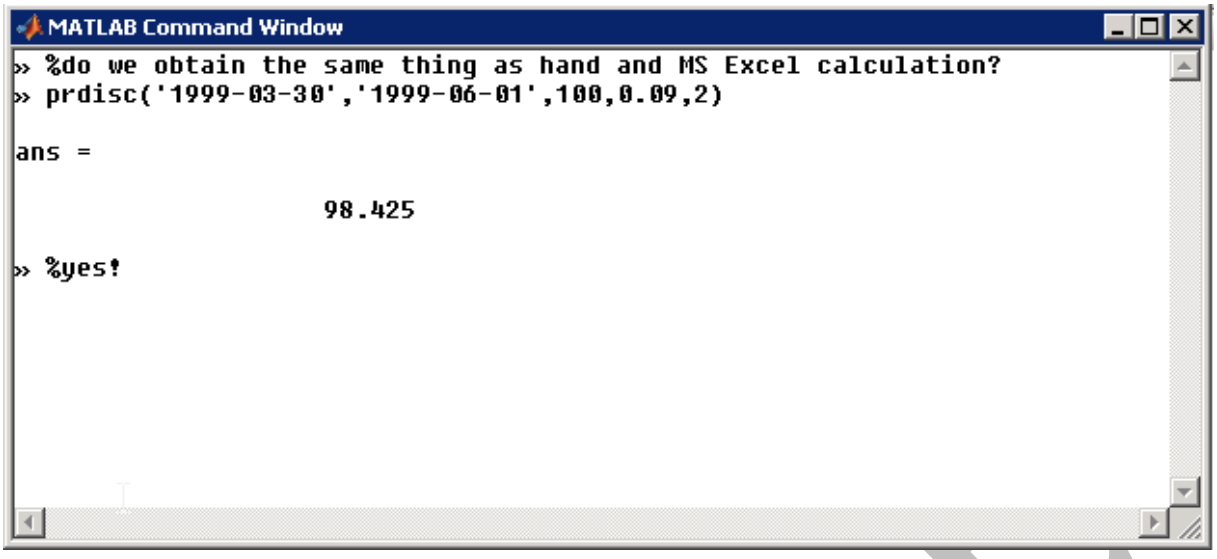
» %this is bond-equivalent. We obtain same thing as calculated by hand
» tbillprice(0.09,'1999-03-30','1999-06-01',3)

ans =

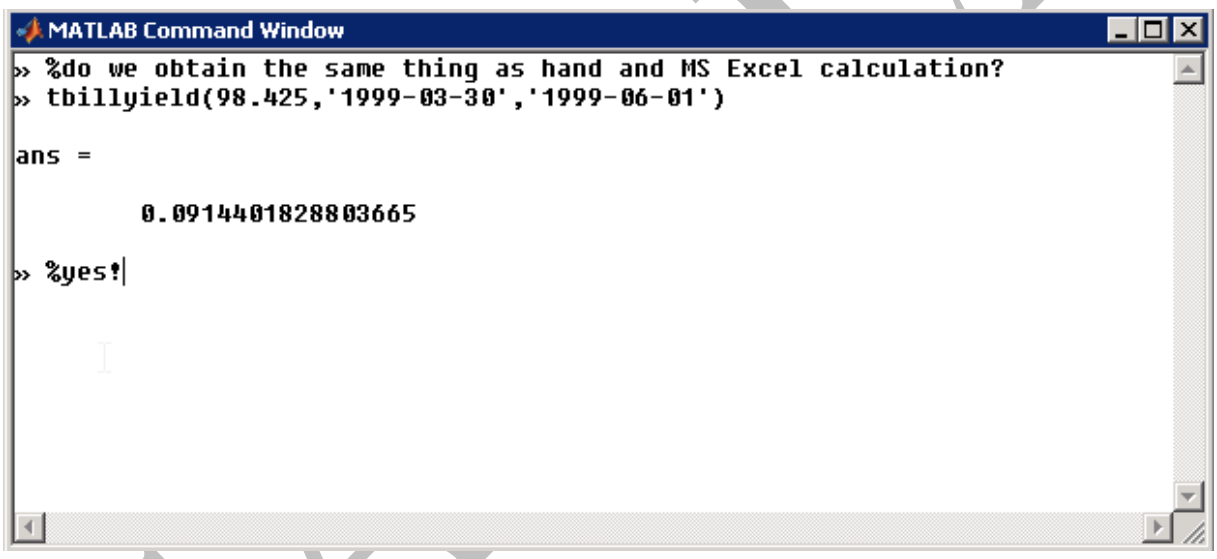
                  98.425

» %same result as in MS Excel (money market price)!
```

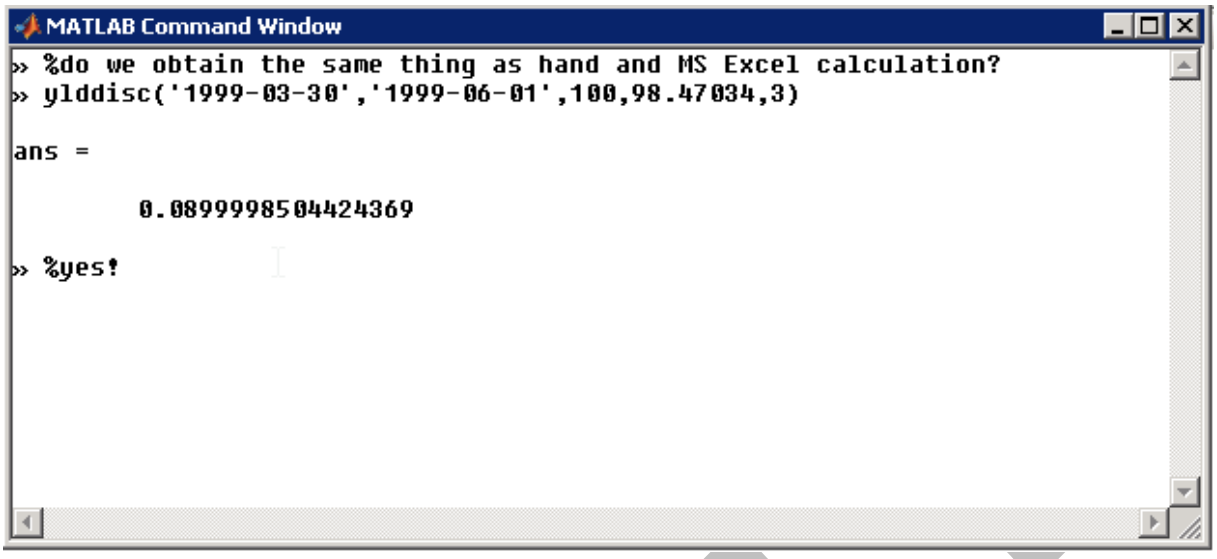
Remember that a treasury bond is by default on 100.-. Pricing a security is based on the same mathematical rules but with a different nominal:



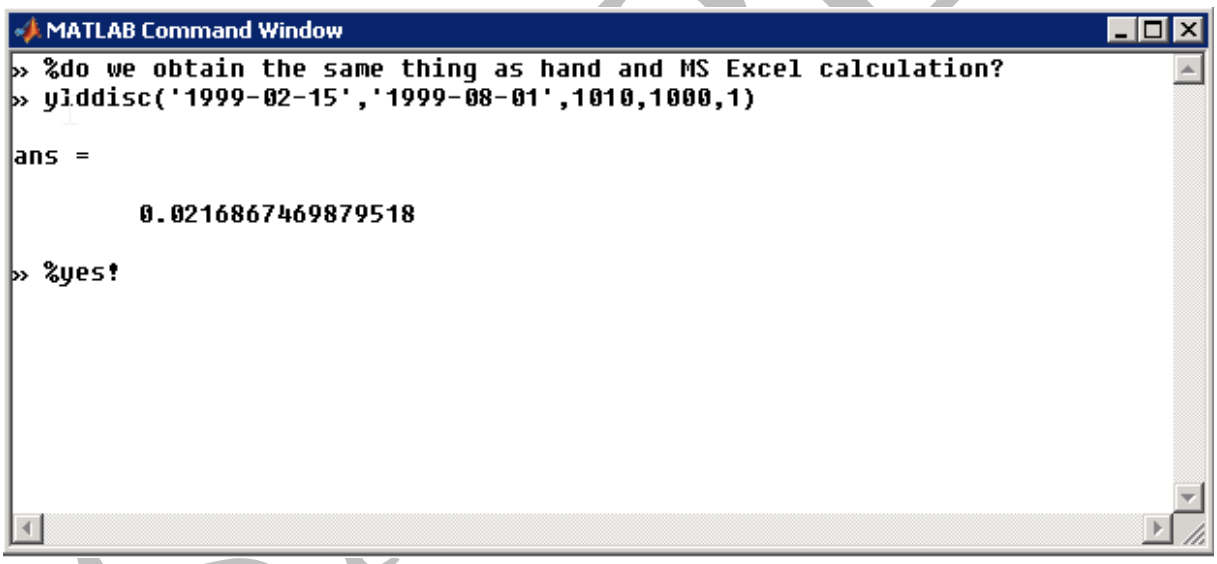
Yield of a treasury bond (TBILLYIELD)



Discount rate of a security (DISC)

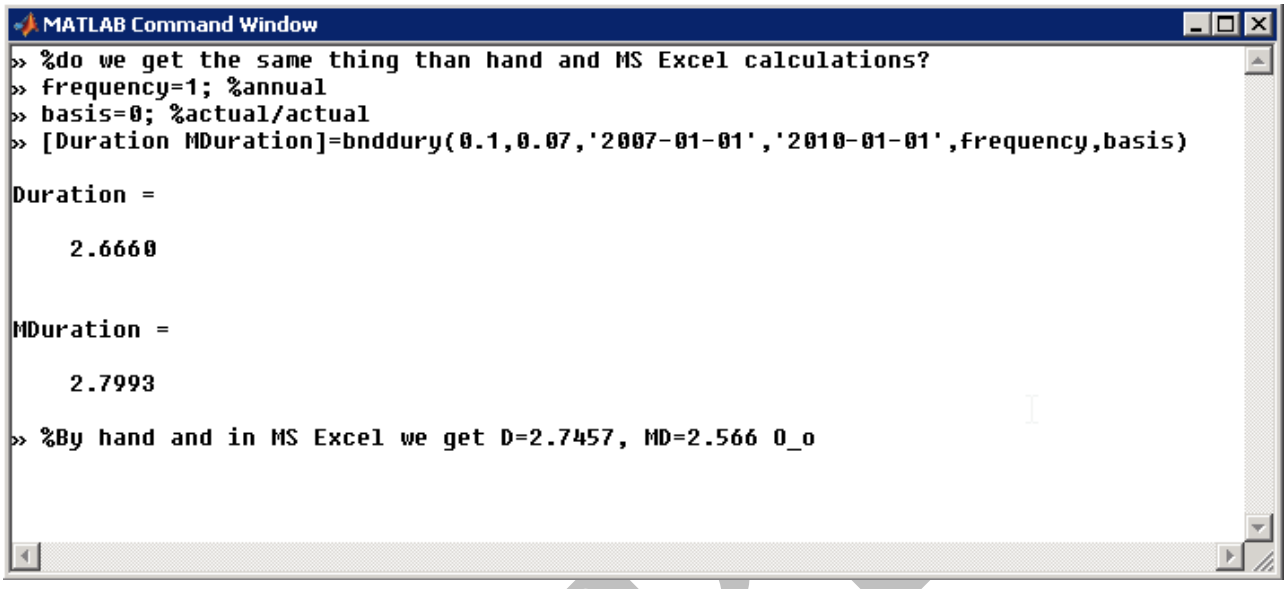


Annual yield for a discounted security (YIELDDISC)



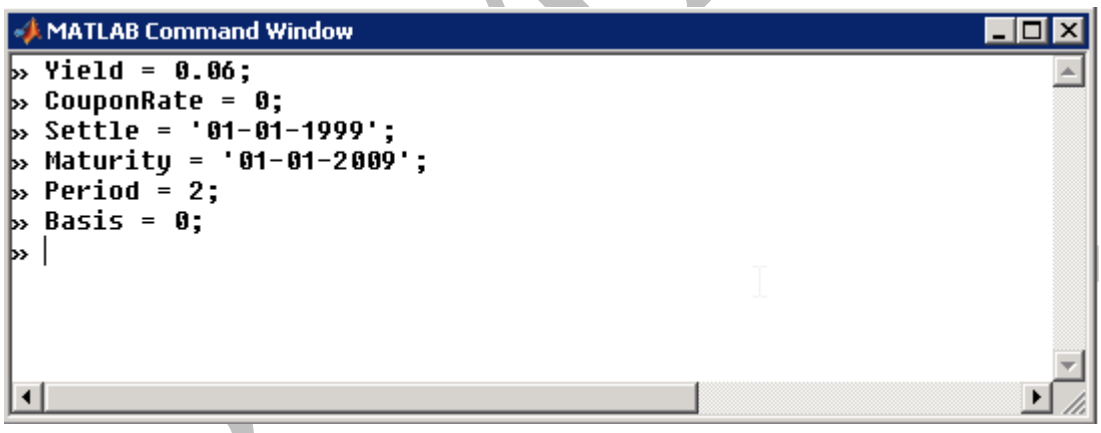
Bond duration, Macaulay duration and convexity

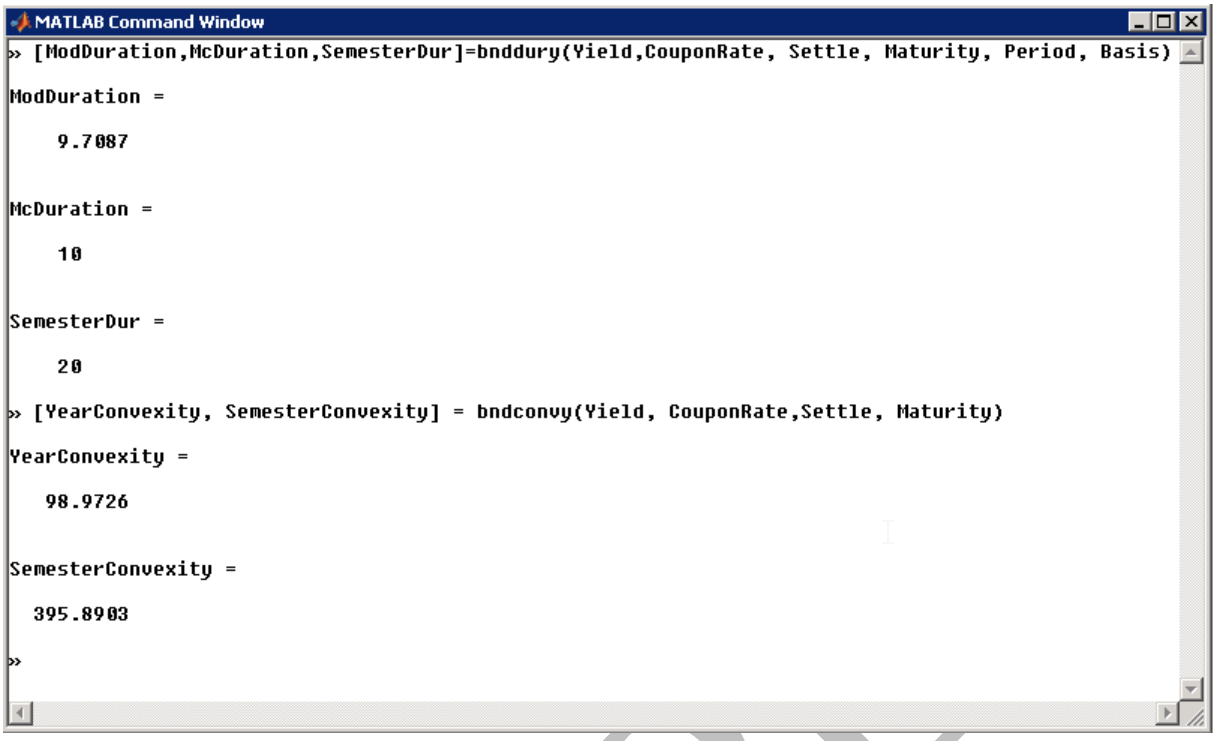
Do we get the same results as the calculations made by hand in Microsoft Excel and with the special native corresponding Microsoft Excel function?:



And in comparison with the calculations made by hand in the theoretical course:

First we define the values:



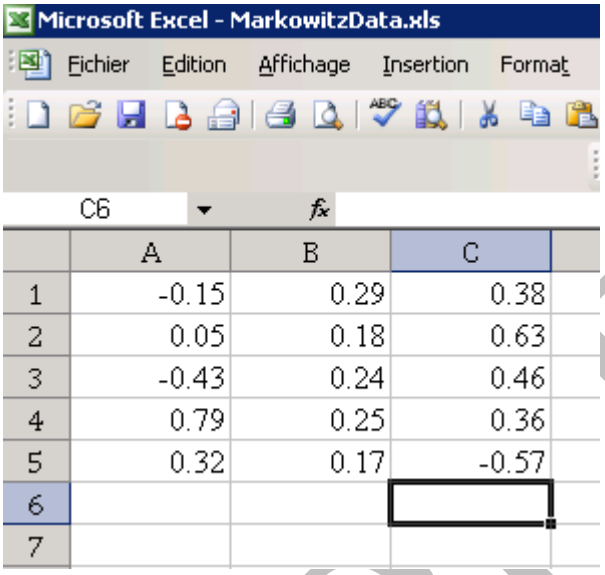


Yes this all is good!

SAMPLE

Naive Markowitz portfolio models

As in the theoretical course we start from the following data of assets:



	A	B	C
1	-0.15	0.29	0.38
2	0.05	0.18	0.63
3	-0.43	0.24	0.46
4	0.79	0.25	0.36
5	0.32	0.17	-0.57
6			
7			

Optimal short-long portfolio (without transaction costs)

To run the naïve Markowitz portfolio without transaction cost the script is the following:

EDITOR PUBLISH VIEW

```
%%Global Minimum Variance Markowitz Model (short sales allowed/unconstrained)
clear all;close all;clc
returns = xlsread('E:\MarkowitzData.xls');
ExpReturns = mean(returns);
c = cov(returns);
StdDev = std(returns);

AssetBounds = [-1 -1 -1 ;1 1 1];
[PortRisk,PortReturn,PortWts]=frontcon(ExpReturns,c,[],[],AssetBounds);
names = {'T1' 'T2' 'T3'};

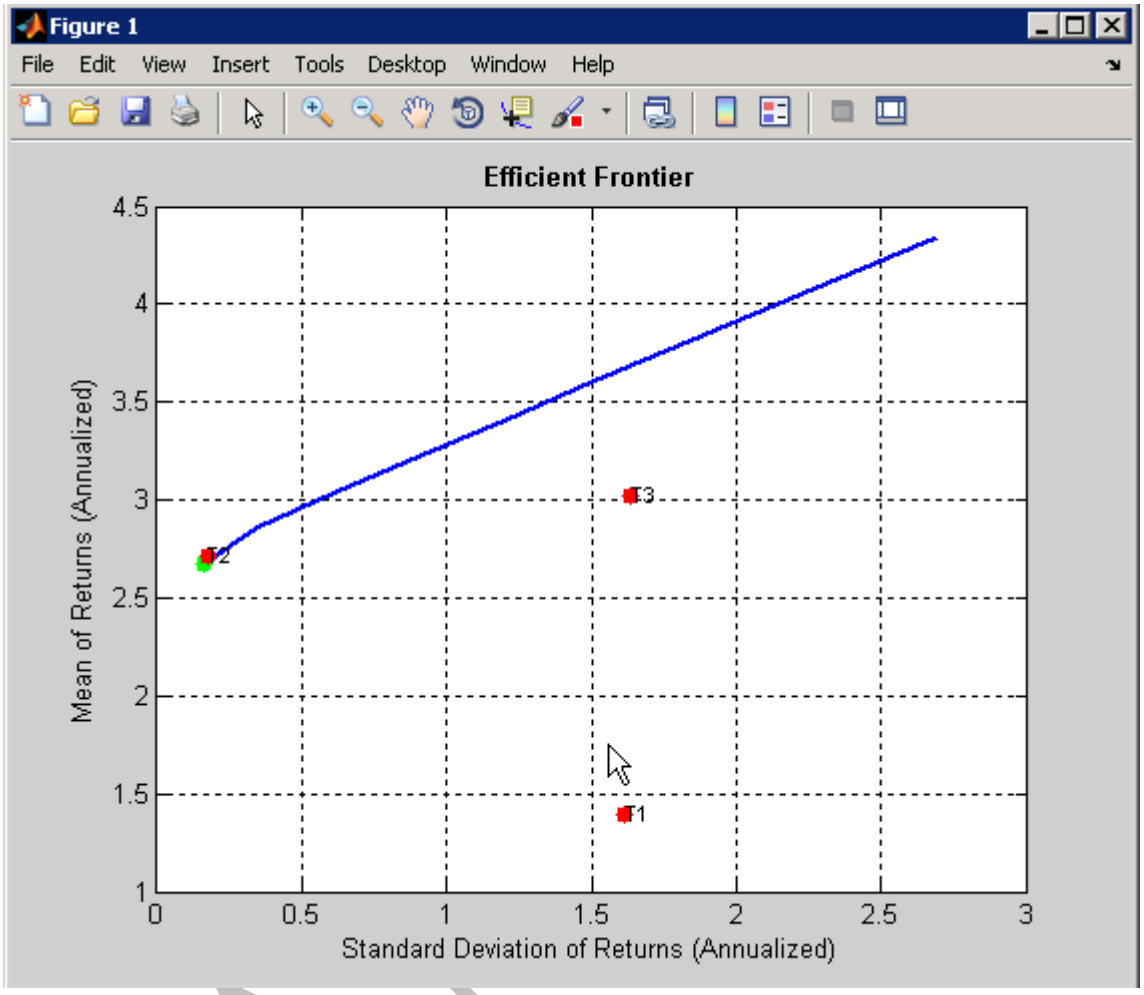
portfolioexamples_plot('Efficient Frontier', ...
    {'line', PortRisk, PortReturn}, ...
    {'scatter',StdDev , ExpReturns, names, 'r'});

RisklessRate=0.001; %can't be null or even NaN (???)
BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
[RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, OverallRisk, ...
    OverallReturn] = portalloc(PortRisk, PortReturn, PortWts,RisklessRate,BorrowRate);
printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')

clf;
portfolioexamples_plot('Efficient Frontier', ...
    {'line', PortRisk, PortReturn}, ...
    {'scatter',RiskyRisk,RiskyReturn},...
    {'scatter',StdDev , ExpReturns, names, 'r'});

RiskyReturn
Variance=RiskyRisk^2
```

We get the following ugly efficient frontier in green the minimum variance portfolio and in red the three initial assets positions:



The corresponding weights, variance and return are:

```
Command Window

Optimal Portfolio =
                          T1           T2           T3
       weight       0.02582      1.00000     -0.02582


RiskyReturn =

    0.2225


Variance =

    0.0021

fx >>
```

Optimal short-long tangent portfolio (without transaction costs)

Ok fine… but let us see is we can reproduce the same example as in the theoretical course with the CML and the chose free-risk rate:

```
clear all;close all;clc;
ExpReturn = [0.116 0.226 0.252];

ExpCovariance = [0.21728   -0.00422   -0.06687
                -0.00422    0.00253    0.01059
                 -0.06687    0.01059     0.22247];
[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,...
ExpCovariance);

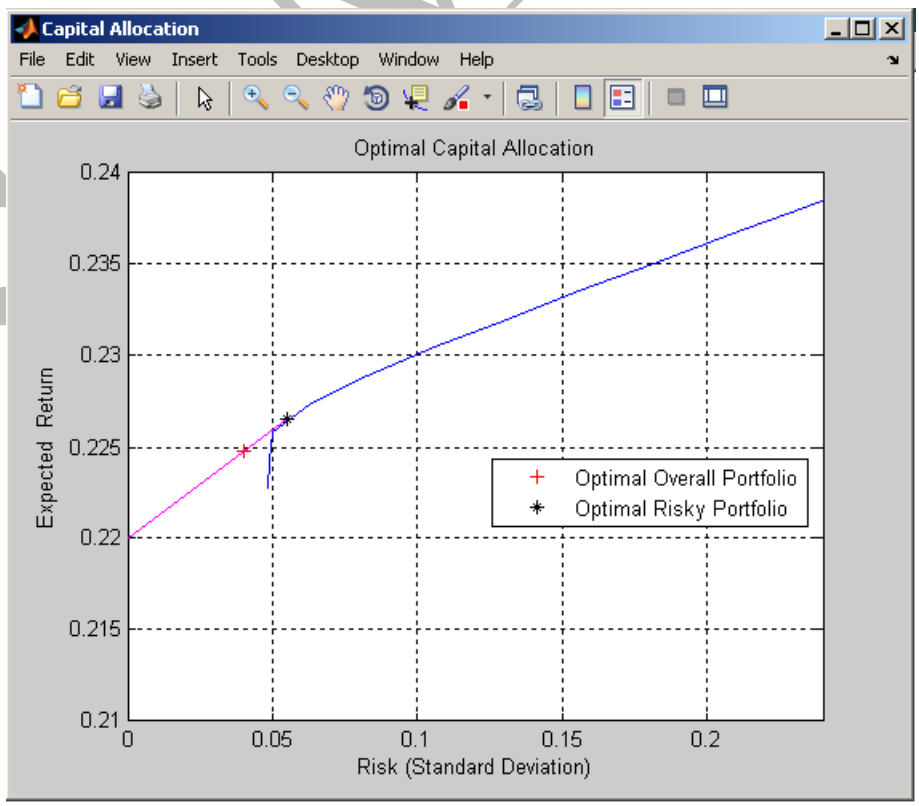
RisklessRate  = 0.22;

[RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, ...
OverallRisk, OverallReturn] = portalloc(PortRisk, PortReturn,...
PortWts, RisklessRate)

NumPorts = 20;

[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,ExpCovariance, NumPorts);
RisklessRate  =  0.22;
portalloc (PortRisk, PortReturn, PortWts, RisklessRate)
axis([0 0.24 0.21 0.24]);
```

And we get:

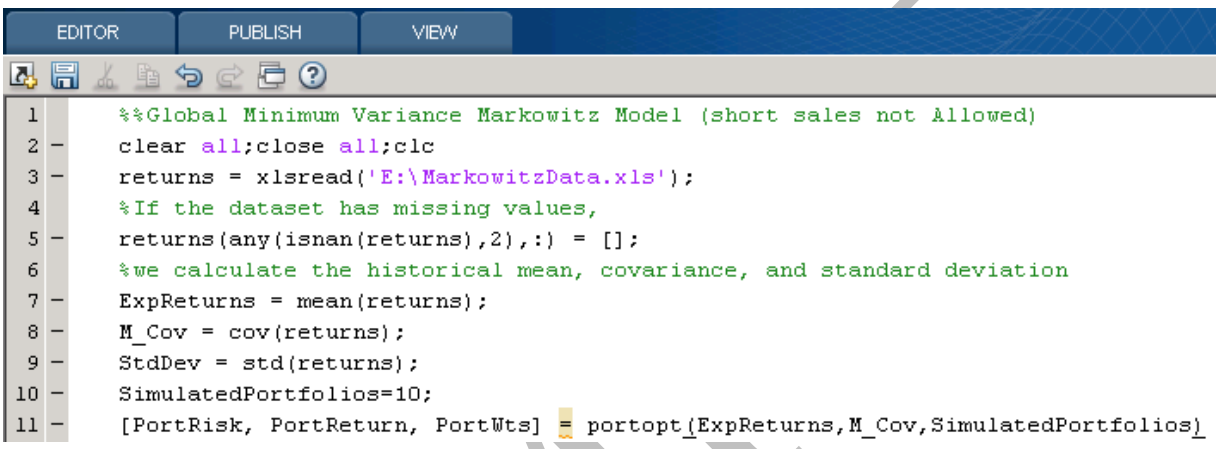


Therefore we see without surprise that we don't get the same results as in Microsoft Excel during the theoretical course but the results are still not too far.

Optimal portfolio with implicit constraints no short sales allowed (without transaction costs)

Now we play with MATLAB to get as in Microsoft Excel a series of returns and variances for various portfolios.

Caution: The results are very sensitive to the number of simulations between 0 and 100 simulations.




```
1   %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2 - clear all;close all;clc
3 - returns = xlsread('E:\MarkowitzData.xls');
4   %If the dataset has missing values,
5 - returns(any(isnan(returns),2),:) = [];
6   %we calculate the historical mean, covariance, and standard deviation
7 - ExpReturns = mean(returns);
8 - M_Cov = cov(returns);
9 - StdDev = std(returns);
10 - SimulatedPortfolios=10;
11 - [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios)
```

We get in the output window:

```
PortRisk =

    0.0483
    0.0503
    0.0879
    0.1389
    0.1929
    0.2481
    0.3037
    0.3595
    0.4155
    0.4717

PortReturn =

    0.2227
    0.2260
    0.2292
    0.2325
    0.2357
    0.2390
    0.2422
    0.2455
    0.2487
    0.2520

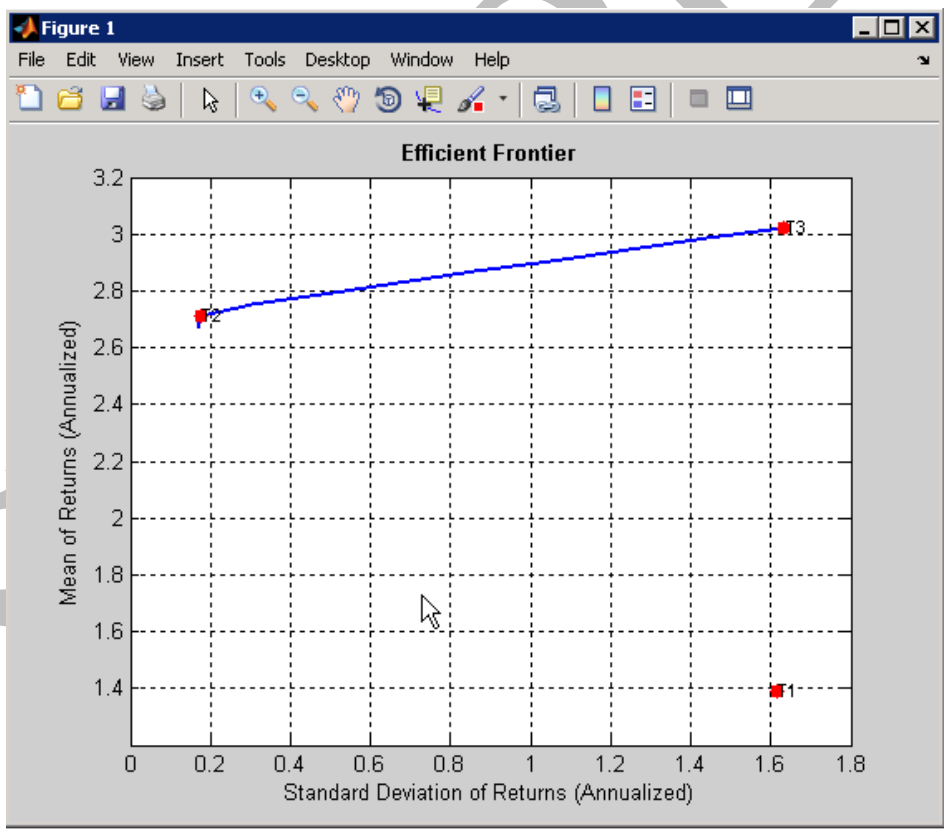
PortWts =

    0.0296    0.9704         0
    0.0000    1.0000         0
         0    0.8751    0.1249
         0    0.7501    0.2499
         0    0.6251    0.3749
         0    0.5001    0.4999
         0    0.3750    0.6250
         0    0.2500    0.7500
         0    0.1250    0.8750
         0         0    1.0000
```

Now by adding a line to the previous script we can plot easily the corresponding efficient frontier:

```
1   %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2   clear all;close all;clc
3   returns = xlsread('E:\MarkowitzData.xls');
4   %If the dataset has missing values,
5   returns(any(isnan(returns),2),:) = [];
6   %we calculate the historical mean, covariance, and standard deviation
7   ExpReturns = mean(returns);
8   M_Cov = cov(returns);
9   StdDev = std(returns);
10  SimulatedPortfolios=10;
11  [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
12  names = {'T1' 'T2' 'T3'};
13  portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
14      {'scatter',StdDev , ExpReturns, names, 'r'});
```

And we get:



Now we add one more constraint:

```
%%Global Minimum Variance Markowitz Model (short sales not Allowed)
clear all;close all;clc
returns = xlsread('E:\MarkowitzData.xls');
%If the dataset has missing values,
returns(any(isnan(returns),2),:) = [];
%we calculate the historical mean, covariance, and standard deviation
ExpReturns = mean(returns);
M_Cov = cov(returns);
StdDev = std(returns);
SimulatedPortfolios=10;
[PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
names = {'T1' 'T2' 'T3'};
portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
    {'scatter',StdDev , ExpReturns, names, 'r'});

RisklessRate=0.001; %can't be null or even NaN (???)
BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
[RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
    = portalloc(PortRisk, PortReturn, PortWts, RisklessRate,BorrowRate);
printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')
```

We get the weights corresponding to the tangent portfolio:

```
Command Window

Optimal Portfolio =
                          T1           T2           T3
        weight       0.01627      0.98373            0

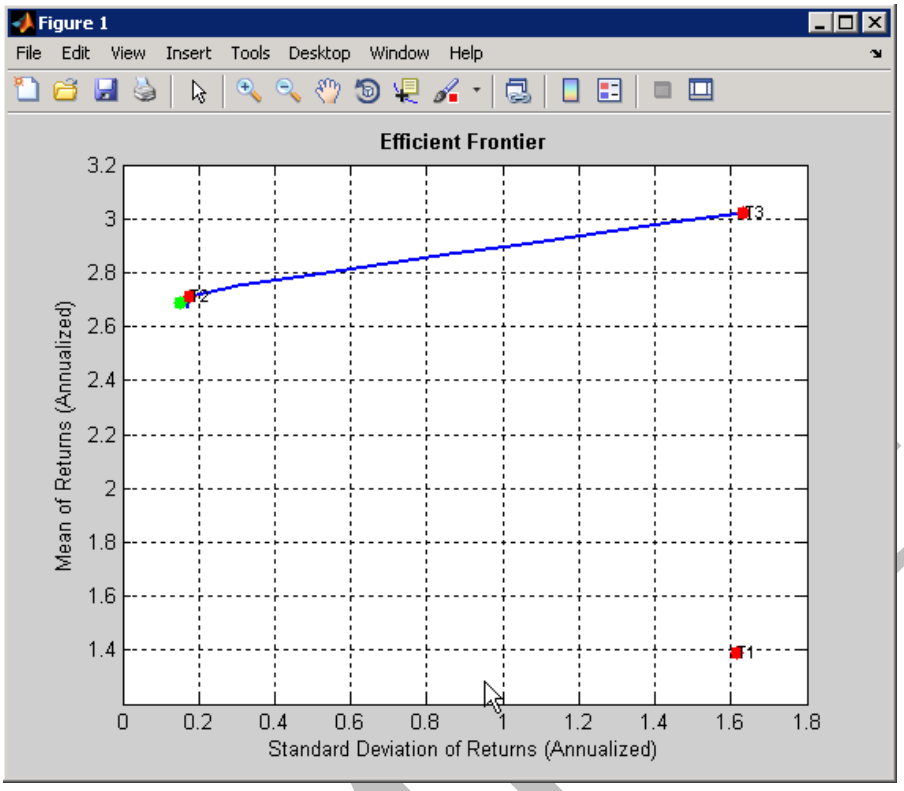
>>
```

If we want to plot all the stuffs:

```
%%Global Minimum Variance Markowitz Model (short sales not Allowed)
clear all;close all;clc
returns = xlsread('E:\MarkowitzData.xls');
%If the dataset has missing values,
returns(any(isnan(returns),2),:) = [];
%we calculate the historical mean, covariance, and standard deviation
ExpReturns = mean(returns);
M_Cov = cov(returns);
StdDev = std(returns);
SimulatedPortfolios=10;
[PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
names = {'T1' 'T2' 'T3'};
portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
    {'scatter',StdDev , ExpReturns, names, 'r'});

RisklessRate=0.001; %can't be null or even NaN (???)
BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
[RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
    = portalloc(PortRisk, PortReturn, PortWts, RisklessRate,BorrowRate);
printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')
clf; %clear figure
portfolioexamples_plot('Efficient Frontier', ...
    {'line', PortRisk, PortReturn}, ...
    {'scatter',RiskyRisk,RiskyReturn},...
    {'scatter',StdDev , ExpReturns, names, 'r'});
```

Therefore we get:



SAMPLE

Optimal portfolio with implicit constraints no short sales allowed (with transaction costs)

Now let us see how we can take into account transaction costs:

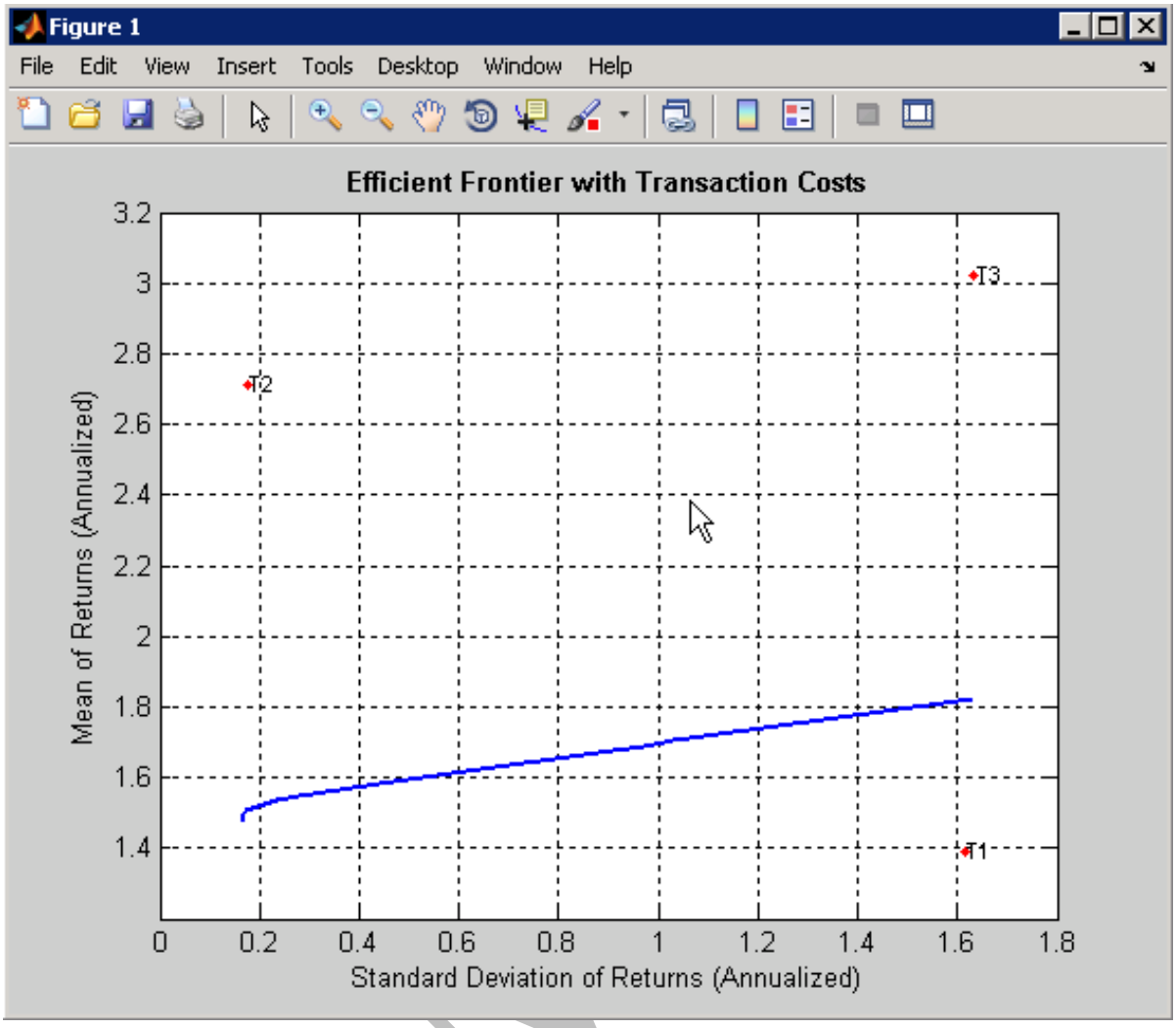
```
clear all;close all;clc
ra=[-0.15,0.05,-0.43,0.79,0.32];
rb=[0.29,0.18,0.24,0.25,0.17];
rc=[0.38,0.63,0.46,0.36,-0.57];
PfM=[0.45,0.61,0.54,0.29,0.12];Rf=repmat(0,1,length(PfM));
data=[ra',rb',rc'];

muMarket=mean(PfM); %market index expected return
sigmaMarket=std(PfM); %market index risk
rf=mean(Rf); %get risk free mean
rfsigma=std(Rf); %get risk free sigma
muAsset=mean(data); %expected return for remaining asset
covAsset=cov(data); %covariance
%create portfolio
port=Portfolio('AssetList',{'T1','T2','T3'},'RiskFreeRate',rf);

%work without Transactions Costs
port=port.setAssetMoments(muAsset, covAsset);
port=port.setDefaultConstraints; %no short sales!

%work with Transactions Costs
%if you put zero you will found the same frontier as before
BuyCost = 0.1; SellCost = 0.3;
q = port.setCosts(BuyCost, SellCost);
qwgt = q.estimateFrontier(30);
[qrsk, qret] = q.estimatePortMoments(qwgt);

% Plot efficient frontiers with gross and net returns
portfolioexamples_plot('Efficient Frontier with Transaction Costs', ...
    {'line', qrsk, qret},...
    {'scatter', sqrt(diag(port.AssetCovar)), port.AssetMean,...
    port.AssetList, '.r'});
```



Get portfolios on the efficient frontier with explicit constraints (without transaction costs)

EDITOR PUBLISH VIEW

```
1   %Obtaining Efficient Portfolios for Target Returns
2   clear all;close all;clc
3   returns = xlsread('E:\MarkowitzData.xls');
4   %If the dataset has missing values,
5   returns(any(isnan(returns),2),:) = [];
6   %Calculate Markowitz Inputs
7   ExpReturns = mean(returns);
8   m_Cov = cov(returns);
9   %for optimization of Global Minimum Variance Portfolio
10  NumPortf=4;
11  PortReturn = []; %must be empty!!
12  AssetBounds = [-1 -1 -1; 1 1 1];
13  [PortRisk, PortReturn, PortWts]=frontcon(ExpReturns,m_Cov, NumPortf, ...
14      PortReturn,AssetBounds)
```

Command Window

```
PortRisk =

    0.0463
    0.2631
    0.5200
    0.7782


PortReturn =

    0.2225
    0.2690
    0.3155
    0.3620


PortWts =

    0.0258    1.0000   -0.0258
   -0.3161    1.0000    0.3161
   -0.6581    1.0000    0.6581
   -1.0000    1.0000    1.0000

fx >>
```

SAM

Equally weighted portfolio (EWP) (without transaction costs)

EDITOR PUBLISH VIEW

```
clear all;close all;clc
ra=[-0.15,0.05,-0.43,0.79,0.32];
rb=[0.29,0.18,0.24,0.25,0.17];
rc=[0.38,0.63,0.46,0.36,-0.57];
PfM=[0.45,0.61,0.54,0.29,0.12];
Rf=repmat(0,1,length(PfM));
data=[ra' rb' rc'];

muMarket=mean(PfM); %market index expected return
sigmaMarket=std(PfM); %market index risk
rf=mean(Rf); %get risk free mean
rfsigma=std(Rf); %get risk free sigma
muAsset=mean(data); %expected return for remaining asset
covAsset=cov(data); %covariance
%create portfolio
port=Portfolio('AssetList',{'T1', 'T2', 'T3'},'RiskFreeRate',rf);
port=port.setAssetMoments(muAsset, covAsset);
%compute the mean and std of equal-weight portfolio returns
port = port.setInitPort(1/port.NumAssets);
[sigmaEwp, muEwp] = port.estimatePortMoments(port.InitPort)

%plot efficient frontier with also Ewp
port = port.setDefaultConstraints; %long only!
pwgt = port.estimateFrontier(30);
[muportfolio, sigmaportfolio] = port.estimatePortMoments(pwgt);
%plot efficient frontier
portfolioexamples_plot('Efficient Frontier', ...
    {'line', muportfolio, sigmaportfolio}, ...
    {'scatter',[sigmaMarket,rfsigma,sigmaEwp],[muMarket,rf,muEwp],...
    {'Market', 'rf', 'Ewp'}}, {'scatter', sqrt(diag(port.AssetCovar)),...
    port.AssetMean, port.AssetList, '.r'});
```

```
Command Window

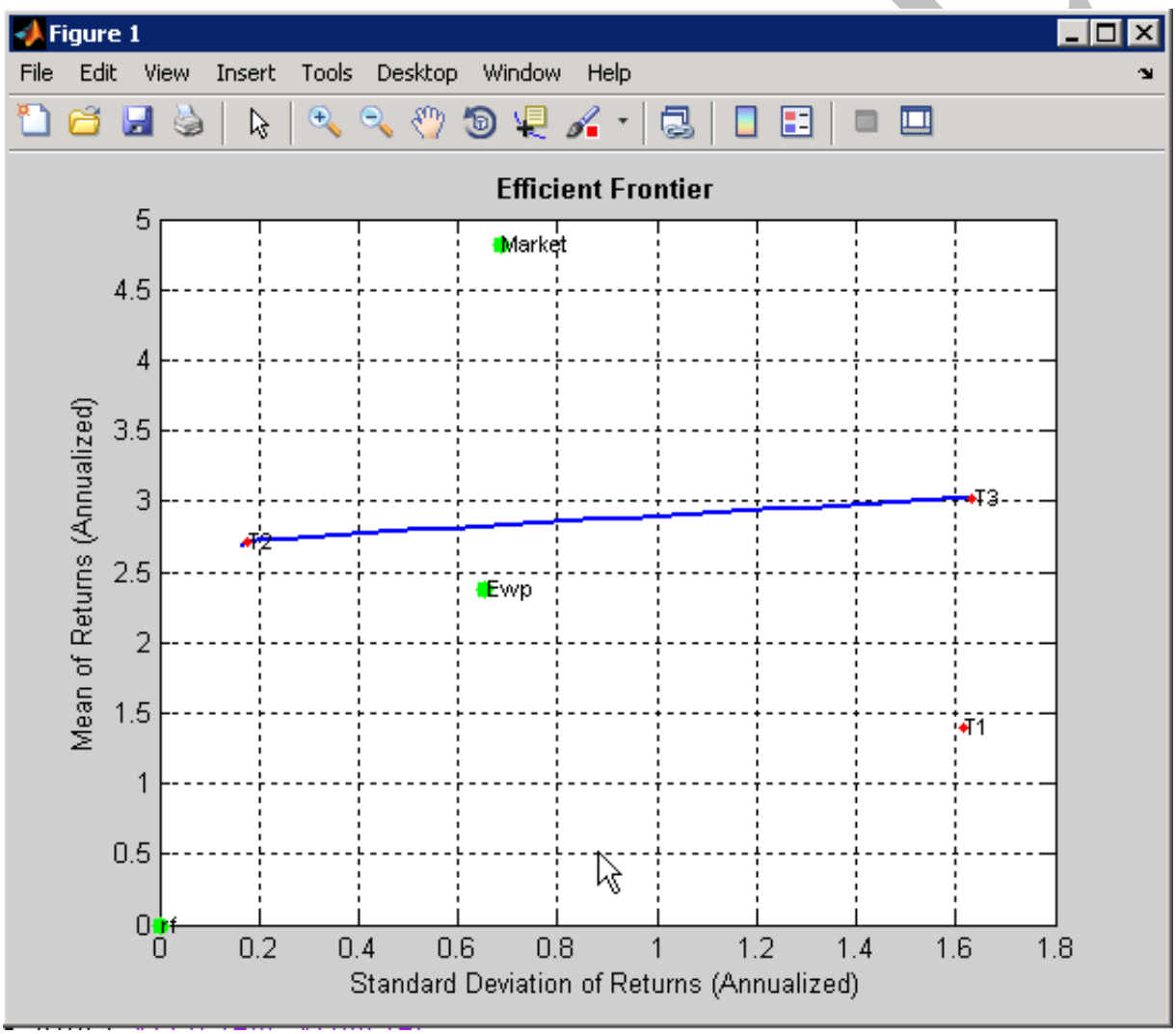
sigmaEwp =

    0.1889

muEwp =

    0.1980

fx >>
```



Maximum Sharpe Ratio (max. diversification) portfolio with implicit constraints (without transaction costs)

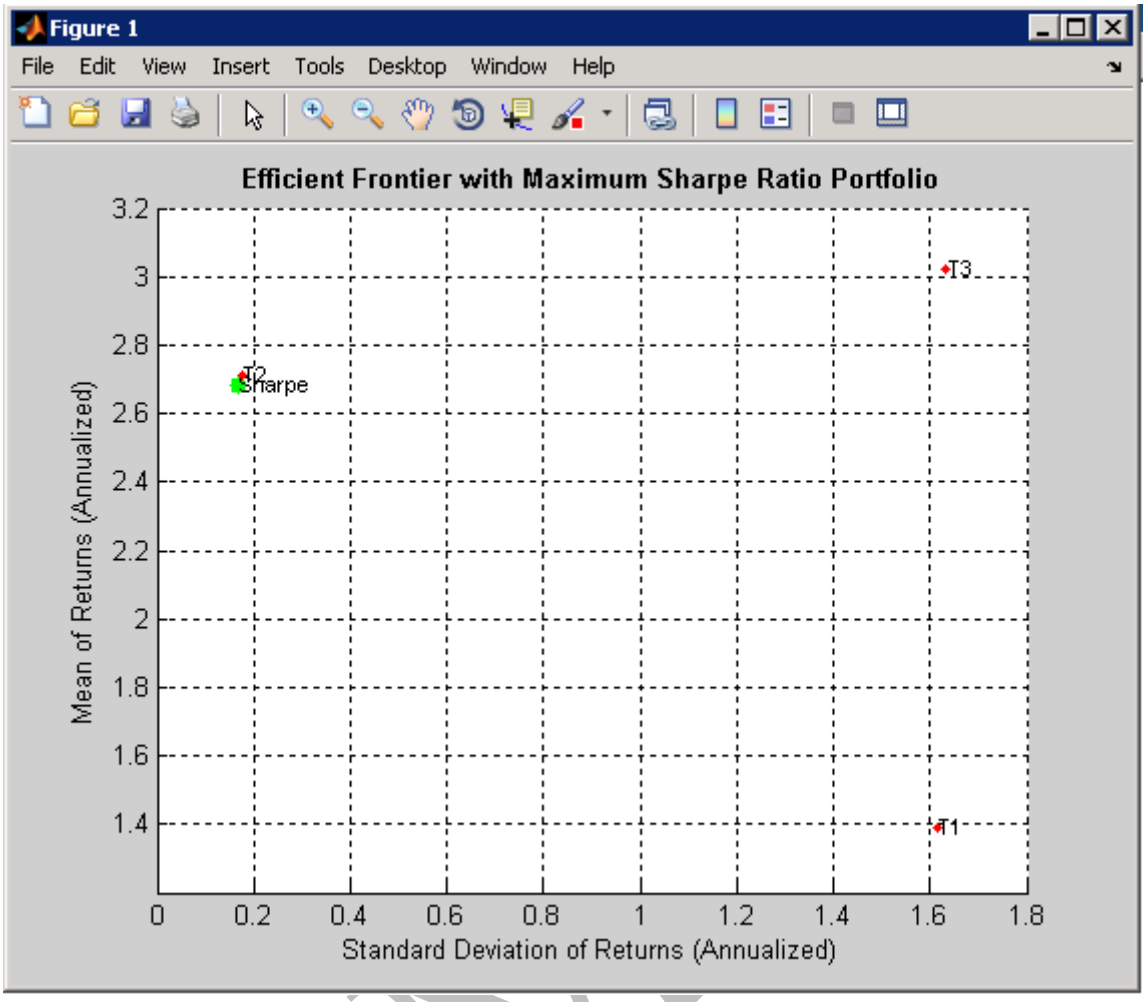
EDITOR PUBLISH VIEW

```
clear all;close all;clc
ra=[-0.15,0.05,-0.43,0.79,0.32];
rb=[0.29,0.18,0.24,0.25,0.17];
rc=[0.38,0.63,0.46,0.36,-0.57];
PfM=[0.45,0.61,0.54,0.29,0.12];Rf=repmat(0,1,length(PfM));
data=[ra',rb',rc'];

muMarket=mean(PfM); %market index expected return
sigmaMarket=std(PfM); %market index risk
rf=mean(Rf); %get risk free mean
rfsigma=std(Rf); %get risk free sigma
muAsset=mean(data); %expected return for remaining asset
covAsset=cov(data); %covariance
%create portfolio
port=Portfolio('AssetList',{'T1','T2','T3'},'RiskFreeRate',rf);

%work without Transactions Costs
port=port.setAssetMoments(muAsset, covAsset);
port=port.setDefaultConstraints; %no short sales!

%Maximize Sharpe Ratio
swgt = port.estimateMaxSharpeRatio;
[srsk, sret] = port.estimatePortMoments(swgt)
% Plot efficient frontier with portfolio that attains maximum Sharpe ratio
portfolioexamples_plot('Efficient Frontier with Maximum Sharpe Ratio Portfolio', ...
    {'scatter', srsk, sret, {'Sharpe'}}, ...
    {'scatter', sqrt(diag(port.AssetCovar)),...
    port.AssetMean, port.AssetList, '.r'});
```



Command Window

```
srsk =

    0.0483


sret =

    0.2233
```

ToDo: Get the weights

Optimal portfolios for target returns and CVaR minimization (without transaction costs)

EDITOR PUBLISH VIEW

```
1     %Obtaining Efficient Portfolios for Target Returns
2     %and CVaR minimization
3 -   clear all;close all;clc
4 -   returns = xlsread('E:\MarkowitzData.xls');
5     %If the dataset has missing values,
6 -   returns(any(isnan(returns),2),:) = [];
7     %Calculate Markowitz Inputs
8 -   ExpReturns = mean(returns);
9 -   m_Cov = cov(returns);
10
11 -  AssetScenarios = mvnrnd(ExpReturns,m_Cov,100000);
12 -  portfolio = PortfolioCVaR;
13 -  portfolio = portfolio.setScenarios(AssetScenarios);
14    % keep the default constraints with nonnegative weights
15 -  portfolio = portfolio.setDefaultConstraints;
16 -  portfolio = portfolio.setProbabilityLevel(0.95);
17 -  targetReturns = [0.2225,  0.05];
18 -  wgt = portfolio.estimateFrontierByReturn(targetReturns);
19 -  printmat(wgt,'Efficient Portfolios for Target Returns',...
20        'T1 T2 T3',num2str(targetReturns))
```

Command Window

```
Warning: One or more target return values are outside the feasible range [ 0.223988,
0.250916 ].
    Will return portfolios associated with endpoints of the range for these values.
> In PortfolioCVaR.estimateFrontierByReturn at 94

Efficient Portfolios for Target Returns =
                    0.2225         0.05
           T1       0.01846       0.01846
           T2       0.98154       0.98154
           T3   7.54824e-22   7.54824e-22

fx >>
```

Optimal portfolios for target risks and CVaR minimization (without transaction costs)

EDITOR PUBLISH VIEW

```
%Obtaining Markwoitz Efficient Portfolios for Target Risks
%by minimizing CVaR
clear all;close all;clc;
returns = xlsread('E:\MarkowitzData.xls');
%If the dataset has missing values,
returns(any(isnan(returns),2),:) = [];
%Calculate Markowitz Inputs
ExpReturns = mean(returns);
m_Cov = cov(returns);

AssetScenarios = mvnrnd(ExpReturns, m_Cov, 1000);
portfolio = PortfolioCVaR;
portfolio = portfolio.setScenarios(AssetScenarios);
% keep the default constraints with nonnegative weights
portfolio = portfolio.setDefaultConstraints;
portfolio = portfolio.setProbabilityLevel(0.95);
targetRisk = [0.00, 0.04];
wgt = portfolio.estimateFrontierByRisk(targetRisk);

printmat(wgt,'Efficient Portfolios for Target Risk',...
    'T1 T2 T3',num2str(targetRisk))
```

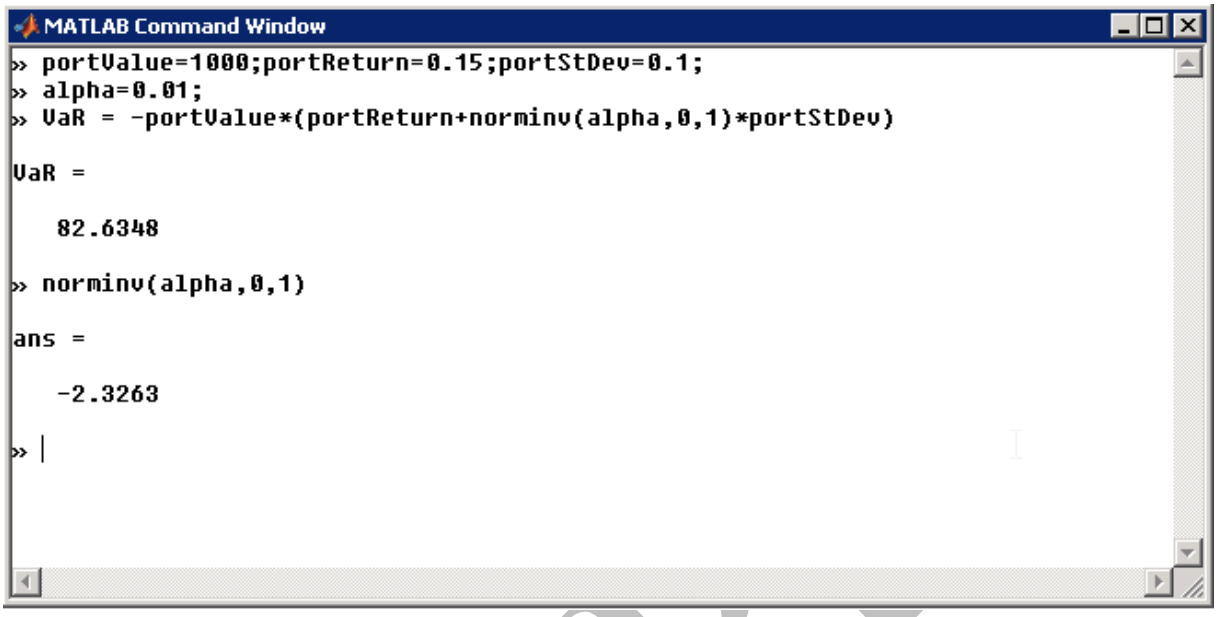
Command Window

```
Efficient Portfolios for Target Risk =
                         0         0.04
           T1   6.31089e-30            0
           T2        0.81921      0.77180
           T3        0.18079      0.22820

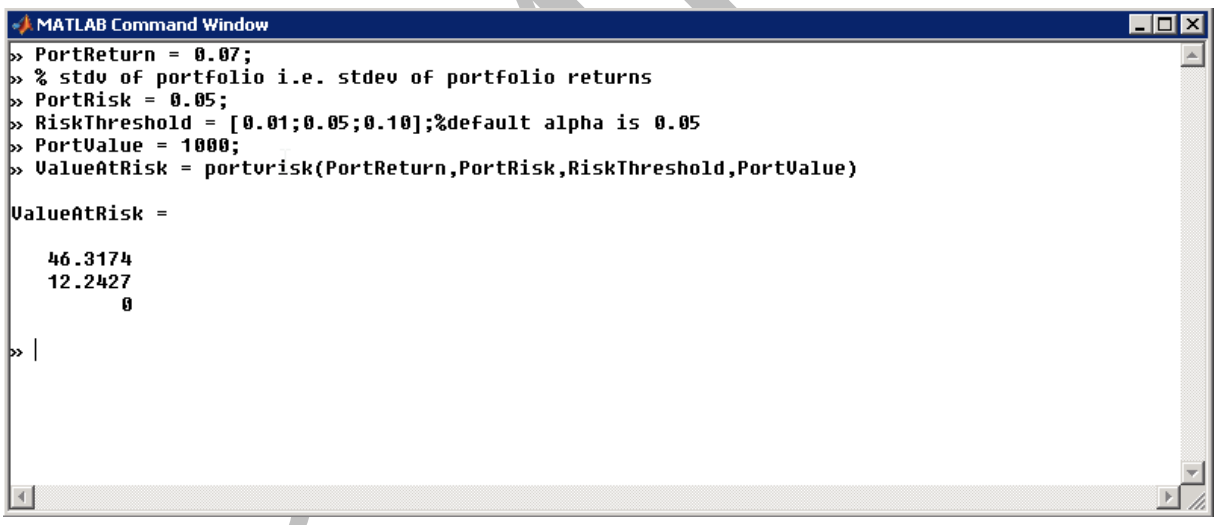
fx >>
```

Value at risk (VaR)

Normal univariate absolute VaR



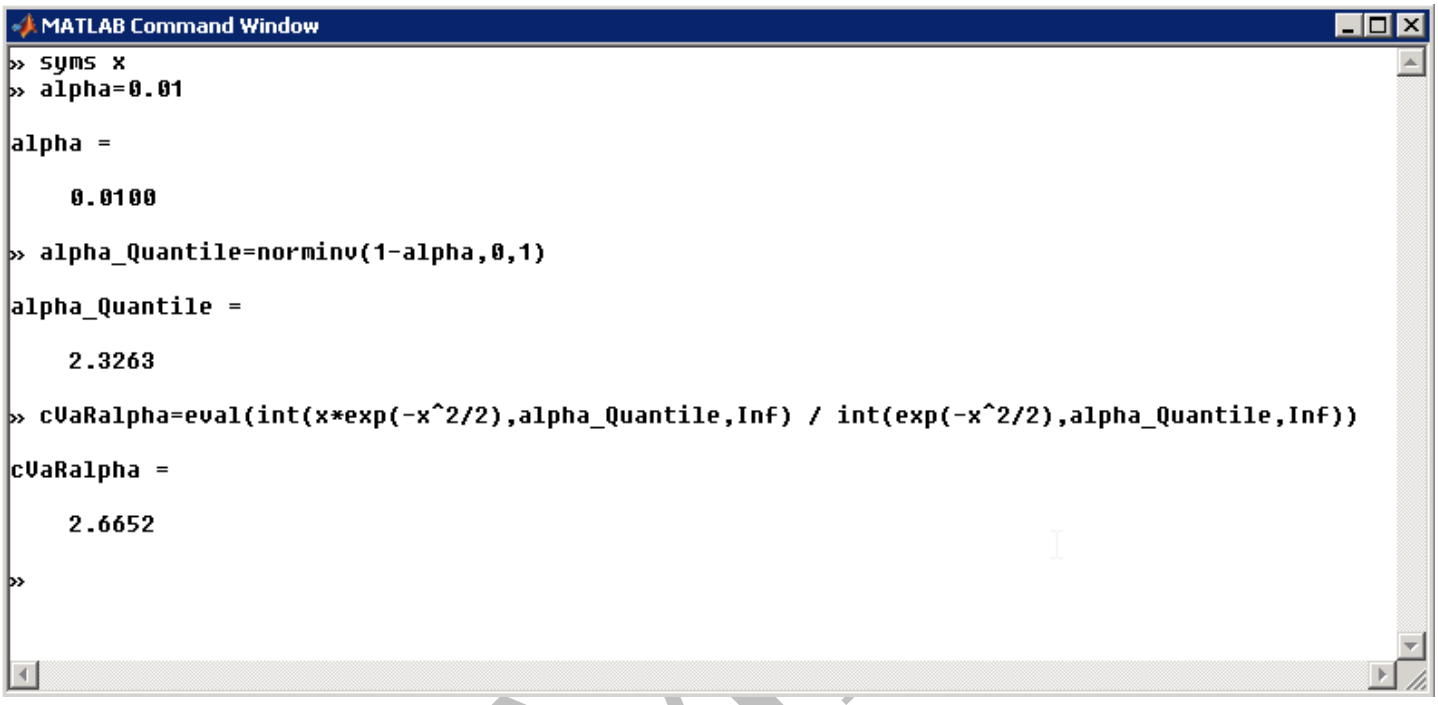
Using MATLAB integrated **portvrisk( )** function we get:



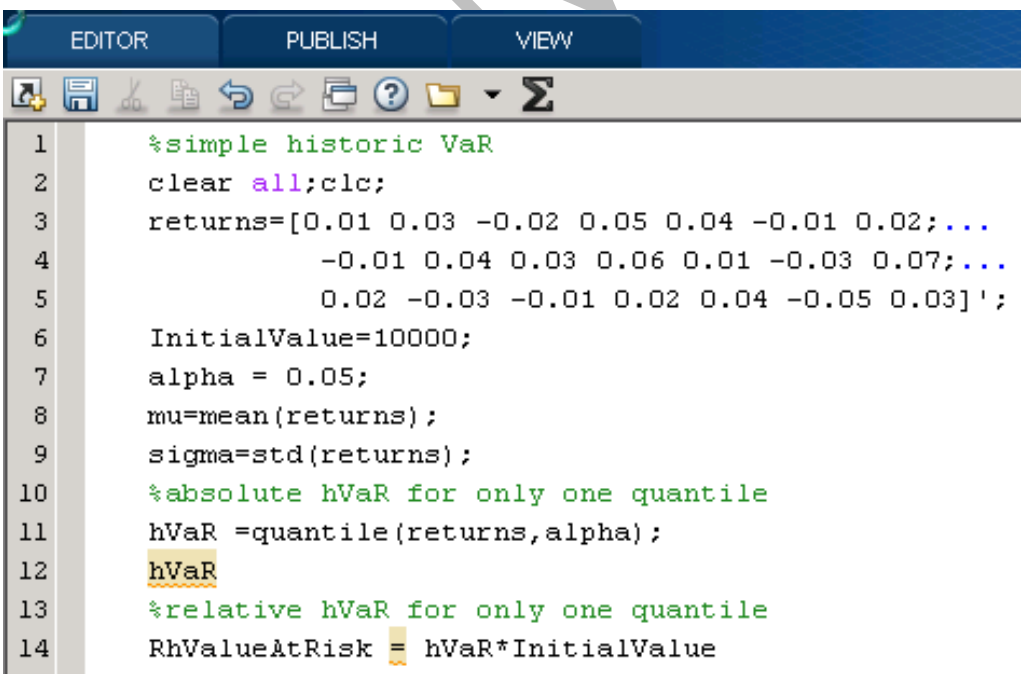
Conditional (absolute) Normal VaR

We need the symbolic Toolbox for this first example:

Soit une VaR suivant une loi Normale qui a été centrée réduite. L'espérance conditionnelle de perte au-delà de 99% (donc au seuil de $\alpha = 1\%$) est alors donnée par:



Historical (absolute and relative) non-parametric VaR



```
Command Window

>> returns

returns =

    0.0100   -0.0100    0.0200
    0.0300    0.0400   -0.0300
   -0.0200    0.0300   -0.0100
    0.0500    0.0600    0.0200
    0.0400    0.0100    0.0400
   -0.0100   -0.0300   -0.0500
    0.0200    0.0700    0.0300


hVaR =

   -0.0200   -0.0300   -0.0500


RhValueAtRisk =

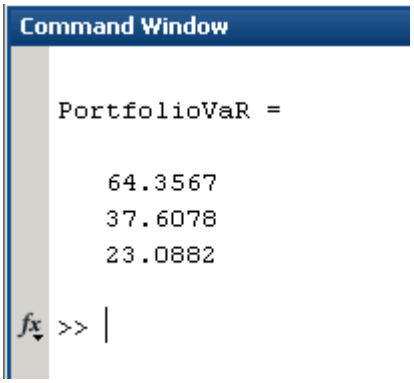
  -200  -300  -500
```

Monte Carlo relative Multivariate Normal VaR

```
EDITOR  PUBLISH  VIEW

1  NumSim=100000; % number of simulation
2  returns=[0.01 0.03 -0.02 0.05 0.04 -0.01 0.02;...
3            -0.01 0.04 0.03 0.06 0.01 -0.03 0.07;...
4            0.02 -0.03 -0.01 0.02 0.04 -0.05 0.03]';
5  mu=mean(returns); %expected return
6  Sigma=cov(returns); %covariance matrix
7  newReturns=mvnrnd(mu,Sigma,NumSim); %simulated returns
8  w=[0.5 0.3 0.2]'; %weigth for each assets in the portfolio
9  Price=[1500 2000 1200]'; %initial price of assets
10 PortValue=Price'*w; %initial value of portfolio
11 %one period forecast thus we use continuous rates
12 PriceSim=repmat(Price,1,NumSim)'.*exp(newReturns); %simulated Price
13 PortSim=PriceSim*w; %simulated Portfolio
14 q=sort(PortSim-PortValue); %simulated profit and loss in ascending order (loss first)
15 alpha=[0.01 0.05 0.1];
16 PortfolioVaR=-q(NumSim*alpha) %tradition to communicate the loss as positive values
```

APLE

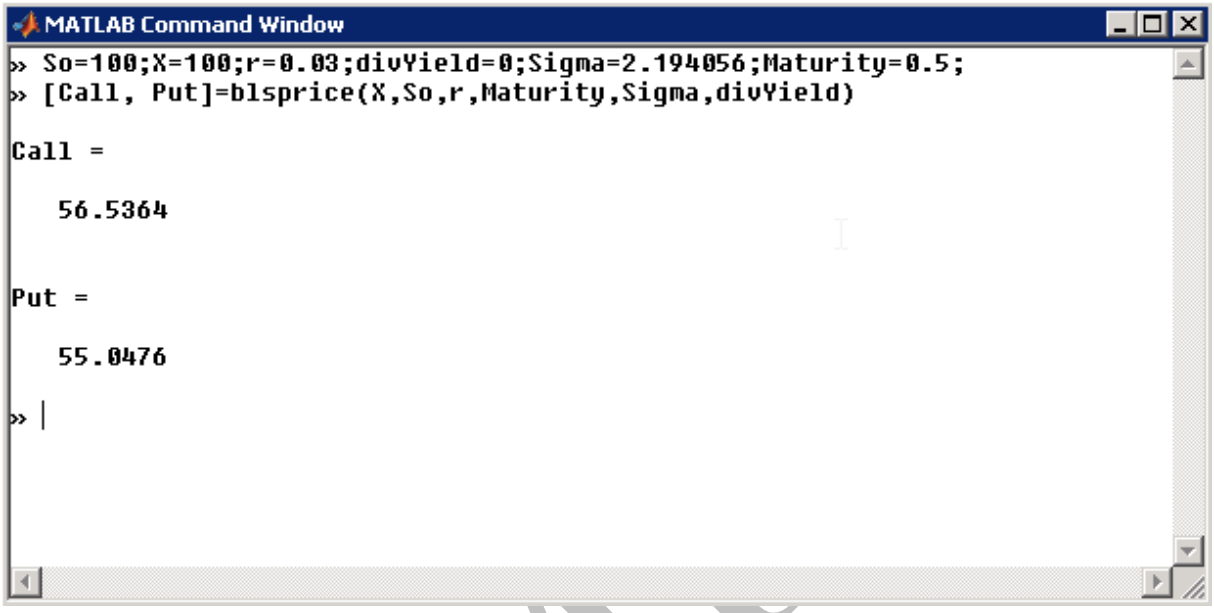


SAMPLE

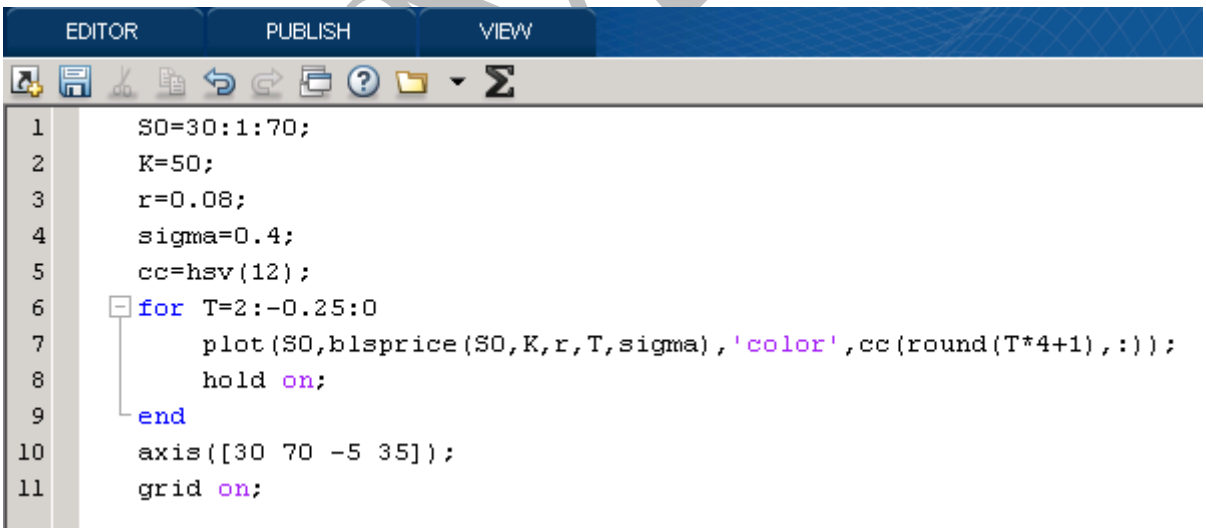
Options Stocks/Futures Pricing & Monitoring

Crazy thing... MATLAB seems to have function only to price European options...!!!

Pricing European Call & Put paying or not dividends (Black & Scholes-Merton Model)



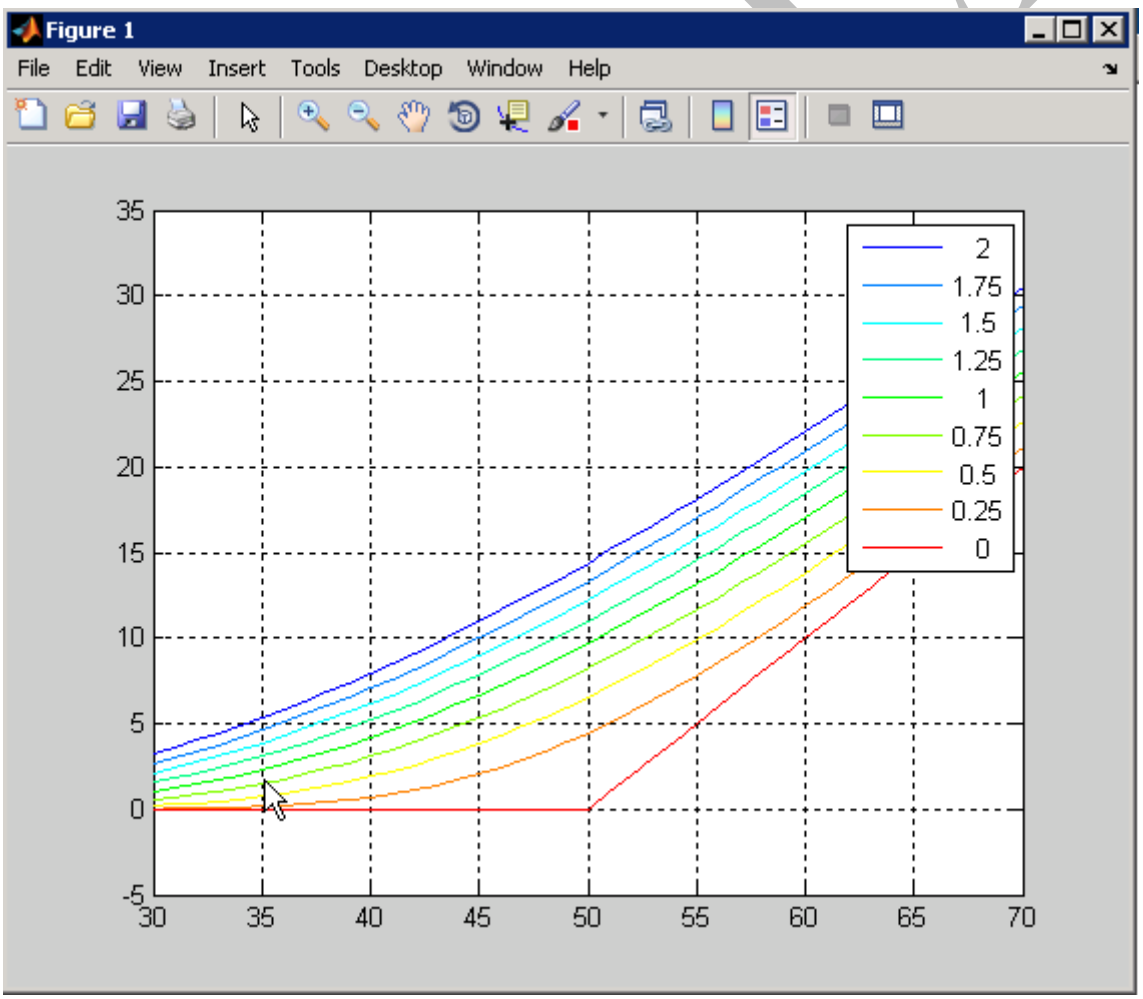
Or another very useful point of view:



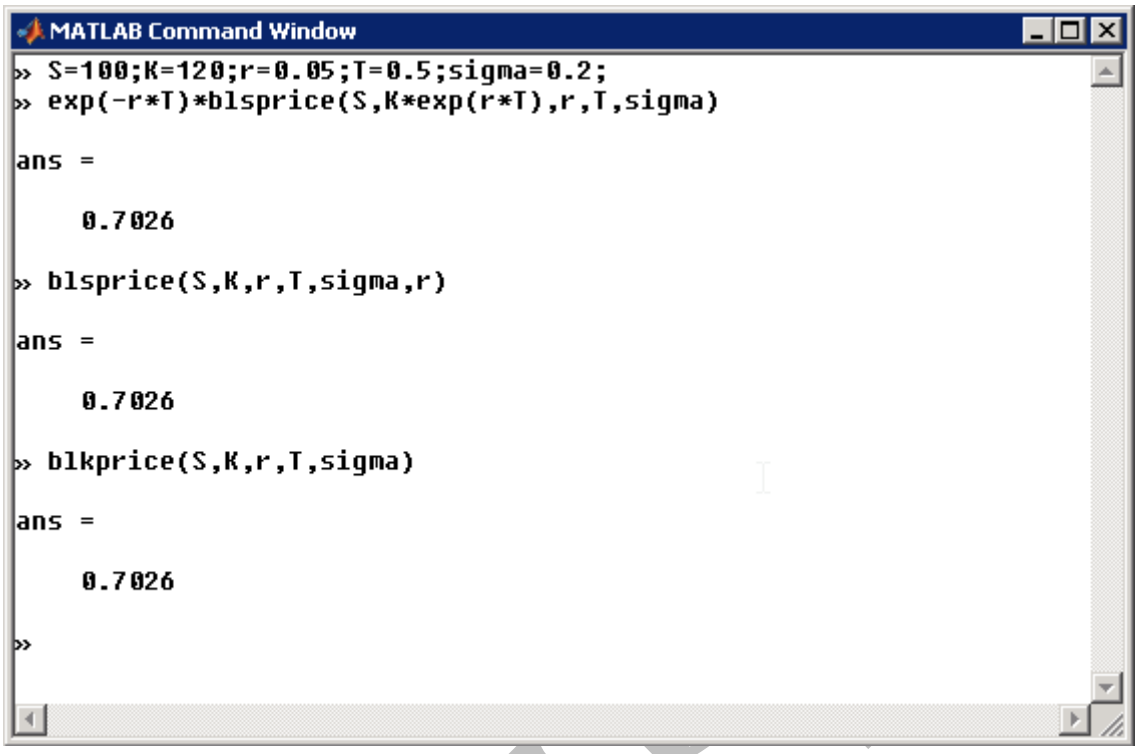
That gives:

```
1   S0=30:1:70;
2   K=50;
3   r=0.08;
4   sigma=0.4;
5   cc=hsv(12);
6   for T=2:-0.25:0
7       plot(S0,blsprice(S0,K,r,T,sigma),'color',cc(round(T*4+1),:));
8       hold on;
9   end
10  axis([30 70 -5 35]);
11  grid on;
12  legend(cellstr(num2str((2:-0.25:0)')))
```

This gives:

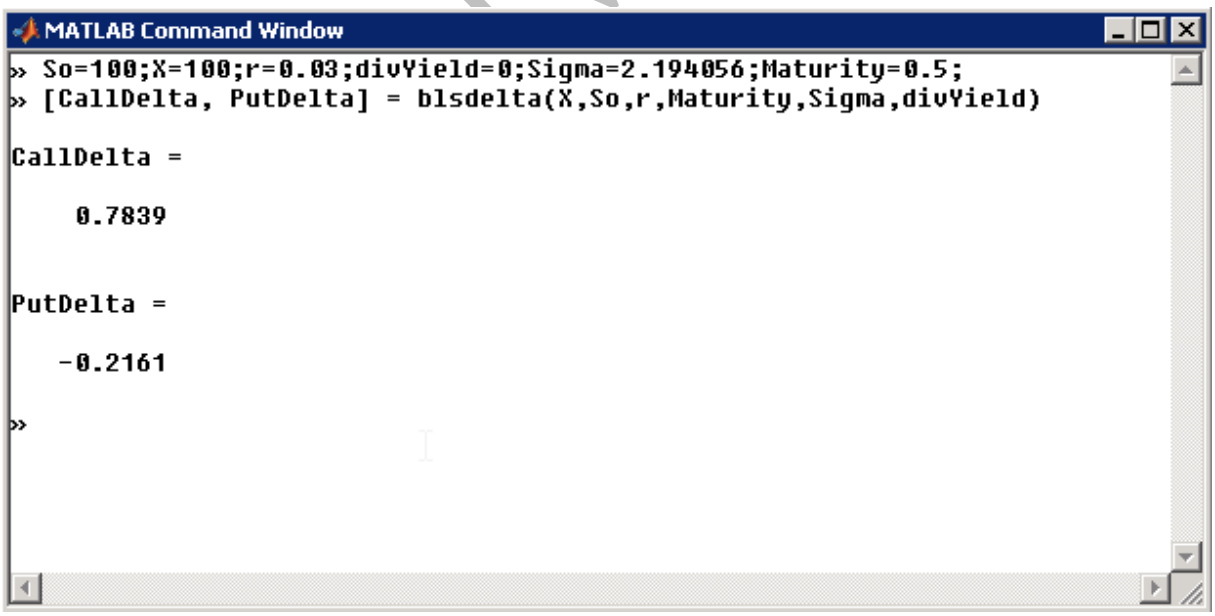


Pricing Future Call (Black's Model)

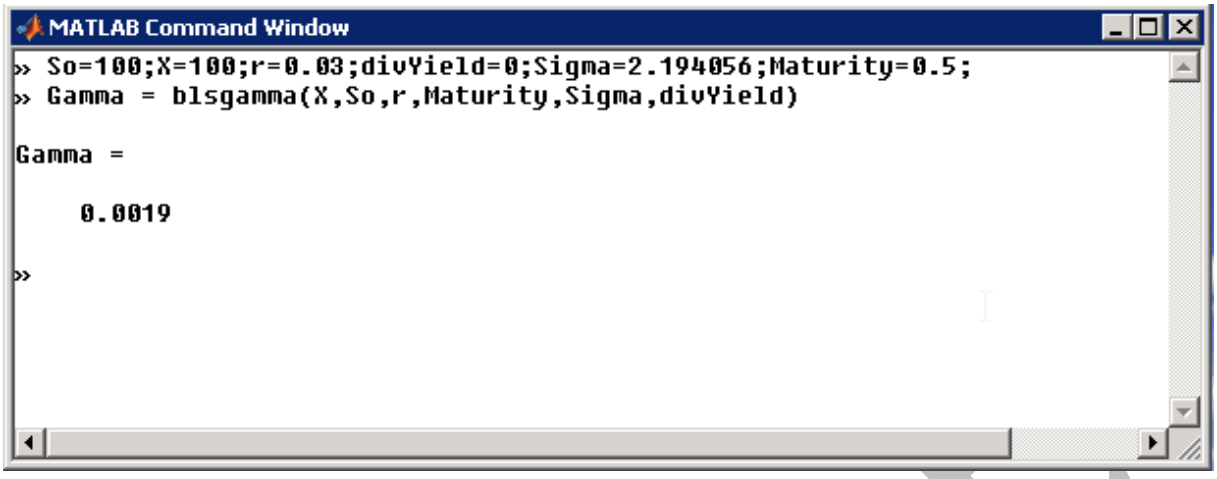


Greeks on European Call & Put paying or not dividends

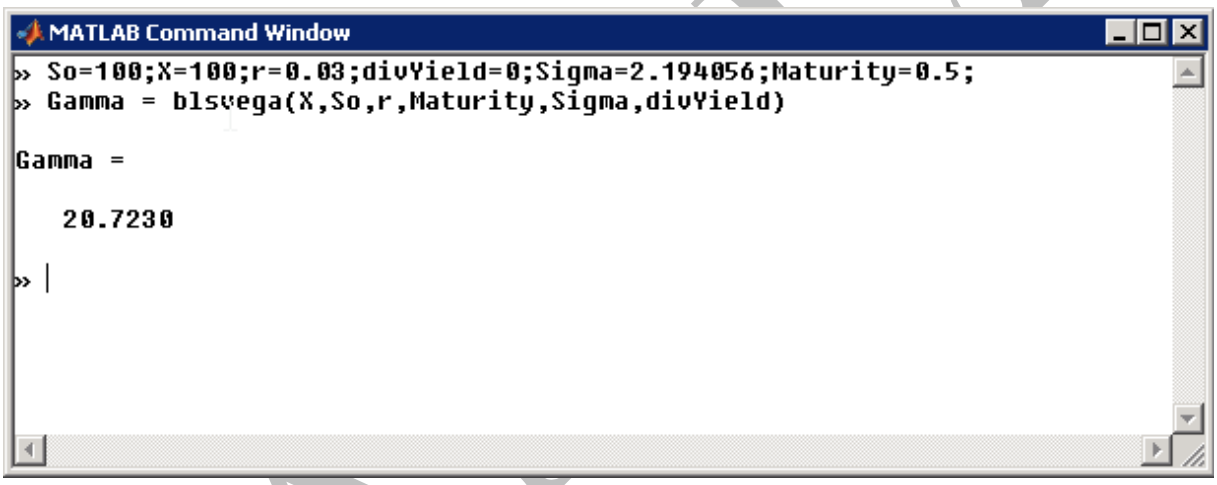
European Option (call/put) Delta



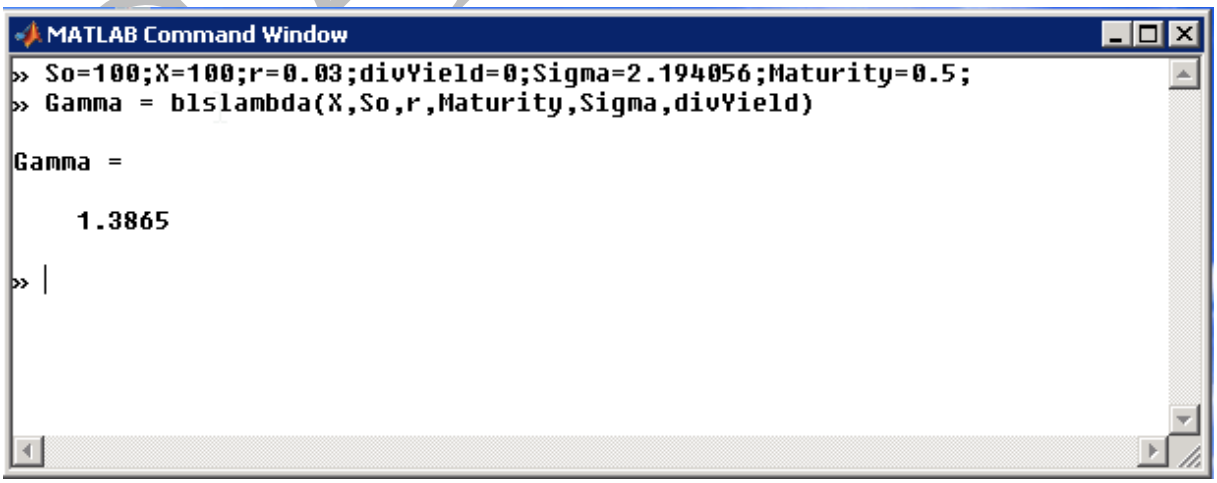
European Option (call/put) Gamma



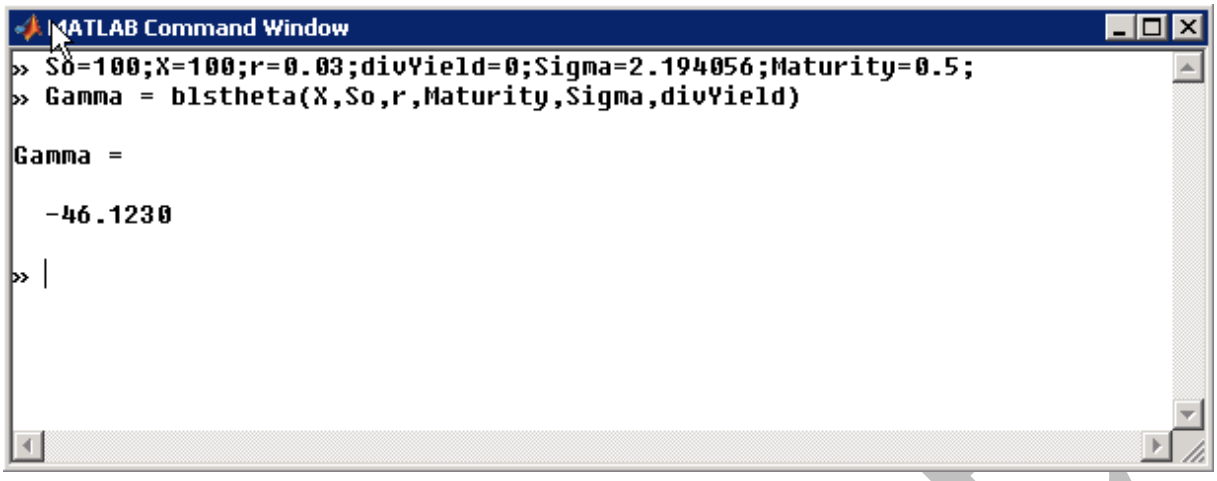
European Option (call/put) Vega



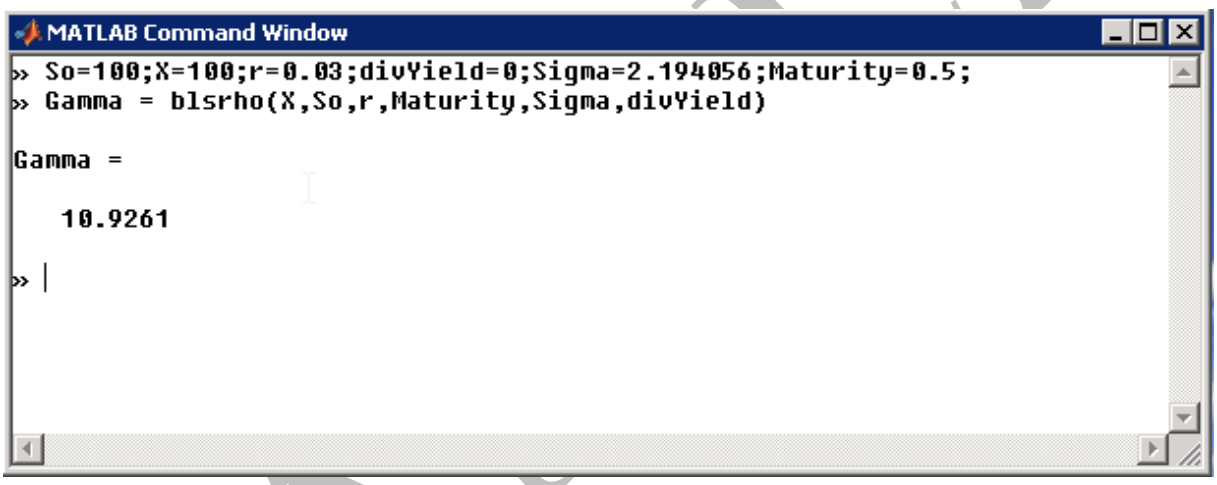
European Option (call/put) Lambda



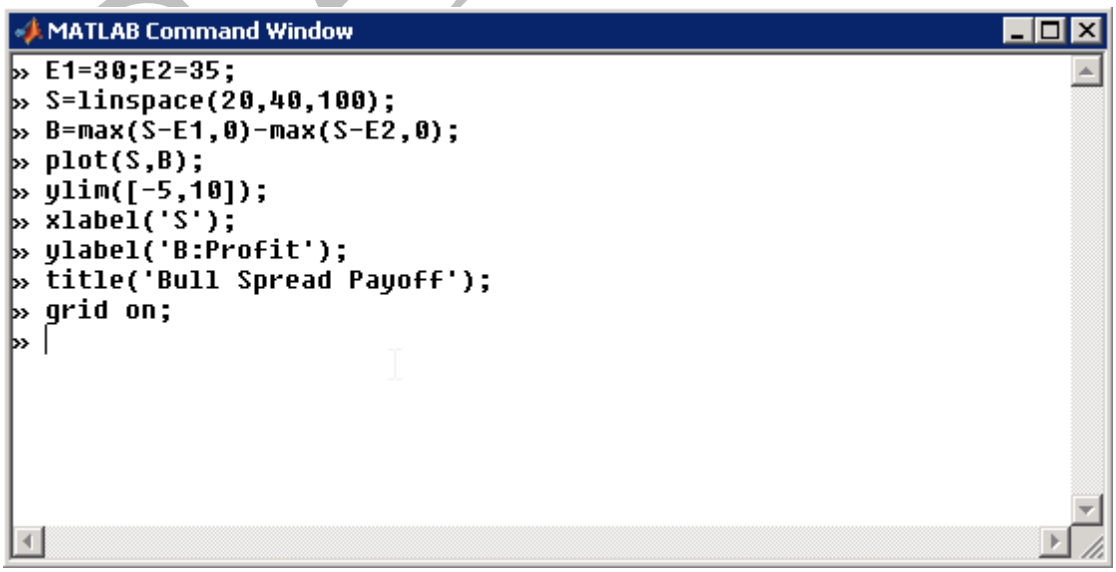
European Option (call/put) theta

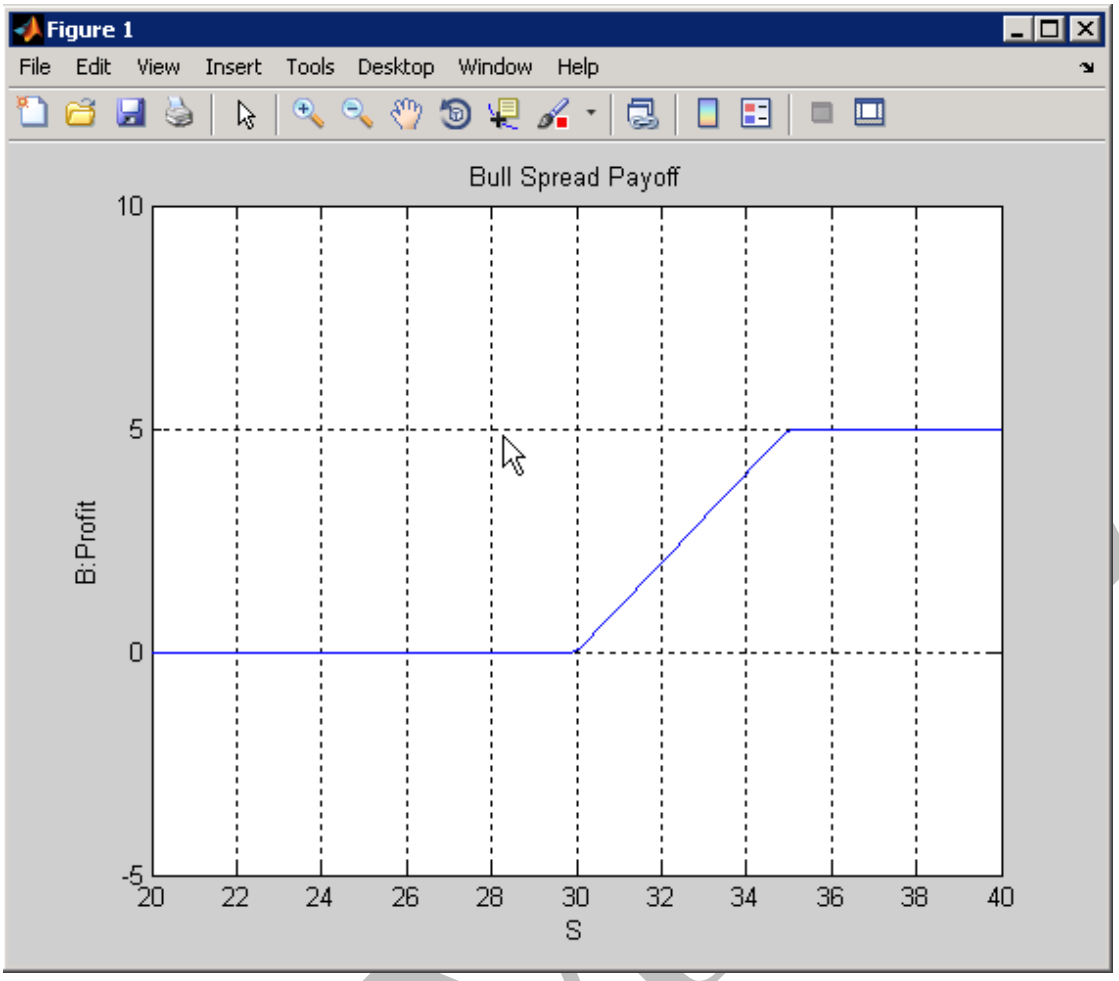


European Option (call/put) Rho



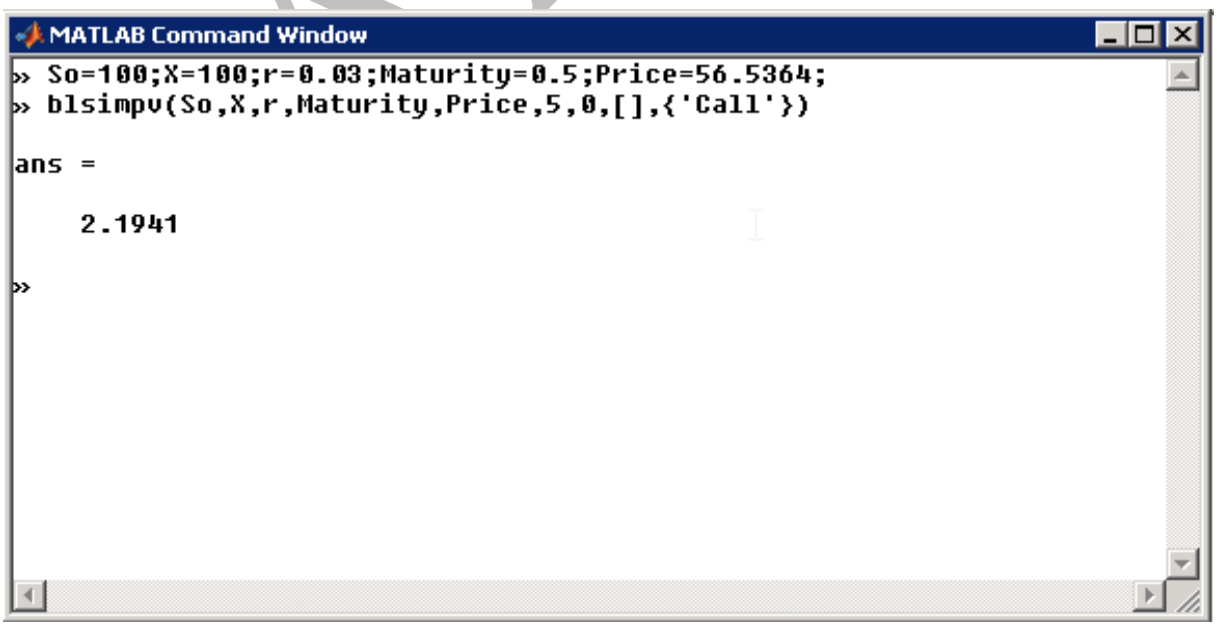
Options trading strategies (payoff diagrams)





Implied volatility surface of an European option

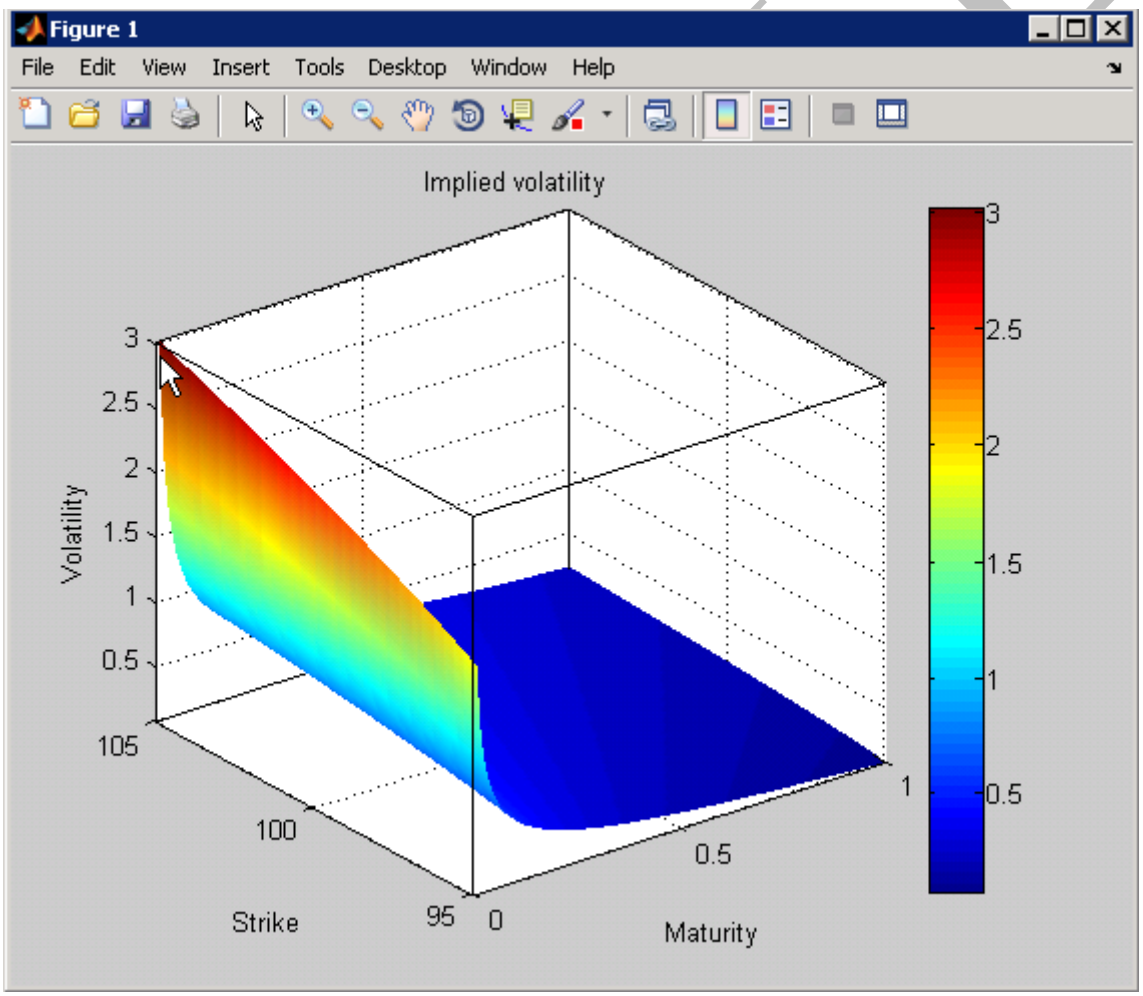
First a one point value estimate:



Now a more general way:

EDITOR PUBLISH VIEW

```
clear all;close all;clc
So=100;X=100;r=0.05;Sigma=0.3;TradingPrice=10;
Maturity=linspace(0,1,100);
Strike=linspace(95,105,100);
[maturity, strike]=meshgrid(Maturity,Strike);
volatility=blsimpv(100,strike,r,maturity,TradingPrice,4,0,[],{'Call'});
surf(maturity,strike,volatility);
xlabel('Maturity');ylabel('Strike');zlabel('Volatility');
colorbar;
box on;
shading interp;
axis tight;
title('Implied volatility');
```



Volatility smile of an European option

EDITOR PUBLISH VIEW

```
1 -    clear all;clc;
2 -    optionData=xlsread('OptionData.xls');
3      % OptionData ::::
4      % the first row represents maturity
5      % the first column, strike prices beginning from 2nd row
6      % the rest are option prices
7 -    optionData
8      % Input parameter
9 -    So=570;
10 -   div=0.0;
11 -   r=0.01;
12 -   PutorCall='Call';
13 -   vec_X=optionData(2:end,1);% get strikes vector
14 -   vec_T=optionData(1,2:end);% get maturnity vector
15 -   priceMatrix=optionData(2:end,2:end);
16 -   Volatility_Smile_And_Surface(PutorCall,So,div,r,vec_X,vec_T,priceMatrix)
```

Microsoft Excel - OptionData.xls

Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?

Arial 10 G

C20

	A	B	C	D	E	F
1	0.00	0.04	0.13	0.20	0.45	0.80
2	660.00	0.45	1.86	3.60	10.50	20.90
3	640.00	1.00	3.30	5.70	14.50	27.90
4	620.00	2.15	5.93	8.90	20.20	34.86
5	600.00	5.30	10.75	14.30	26.81	42.80
6	580.00	11.50	17.30	23.00	36.80	48.73
7	560.00	21.30	28.10	32.80	46.50	60.40
8	540.00	36.60	43.10	45.90	59.50	73.00

Command Window

```
optionData =

          0    0.04    0.13    0.20    0.45    0.80
   660.00    0.45    1.86    3.60   10.50   20.90
   640.00    1.00    3.30    5.70   14.50   27.90
   620.00    2.15    5.93    8.90   20.20   34.86
   600.00    5.30   10.75   14.30   26.81   42.80
   580.00   11.50   17.30   23.00   36.80   48.73
   560.00   21.30   28.10   32.80   46.50   60.40
   540.00   36.60   43.10   45.90   59.50   73.00
```

EDITOR PUBLISH VIEW

```
function Volatility_Smile_And_Surface(PutOrCall,S0,div,r,vec_X,vec_T,priceMatrix)
    NumX = length(vec_X);
    NumM = size(vec_T,2);
    for i = 1:NumM
        T = vec_T(1,i);
        % Loop for the different strike prices
        for iStrike = 1:NumX
                price = priceMatrix(iStrike,i);
                X = vec_X(iStrike);
            try
                % Calculation of the implied volatility
                %ImpliedVolatility(PutOrCall,S0,X,r,T,div,Optionprice)
                implVolM(iStrike,i) = ImpliedVolatility(PutOrCall,S0,X,r,T,div,price);
                % Add nan to the value matrix if the function was not able to
                % solve for the implied volatility.
            catch MExc
                implVolM(iStrike,i) = nan;
            end
        end
    end
    implVolM
    % Compute vector for moneyness
    vec_moneyness = log(S0./vec_X);
    % Plot volatility smile
    figure(1);
    plot(vec_X, implVolM(:,1),'r-','LineWidth',2);
    hold on;
    plot(vec_X, implVolM(:,NumM),'b-','LineWidth',2);
    title('Volatility Smile');xlabel('Strike');ylabel('Volatility');
    % Set the range of axis: [x_min x_max y_min y_max]
    axis([min(vec_X)-10 max(vec_X)+10 min(min(implVolM))-0.1  max(max(implVolM))+0.1]);
    legend('Implied Volatility for short Maturity',...
    'Implied Volatility for long Maturity','location','NorthWest');
    % Plot volatility surface
    figure(2);
    surf(vec_moneyness,vec_T,implVolM');
    title('Volatility Surface');
    xlabel('Moneyness');ylabel('Maturity (in years)'); zlabel('Volatility');
    % Set the y-axis to evolve backwards
    set(gca,'YDir','reverse');
    colorbar;shading interp;
end
```

EDITOR PUBLISH VIEW

```
function [ IV ] = ImpliedVolatility(PutOrCall,S0,X,r,T,div,Optionprice )
    options = optimset('fzero');
    %TolX, 1e-7: tolerance level
    options = optimset(options, 'TolX', 1e-7);
    func = @(sigma)diffbls2price(PutOrCall,S0,X,r,sigma,T,div,Optionprice);
    % [0.001 5] interval for volatility. To change if necessary!
    IV = fzero(func,[0.001 5],options);
end
```

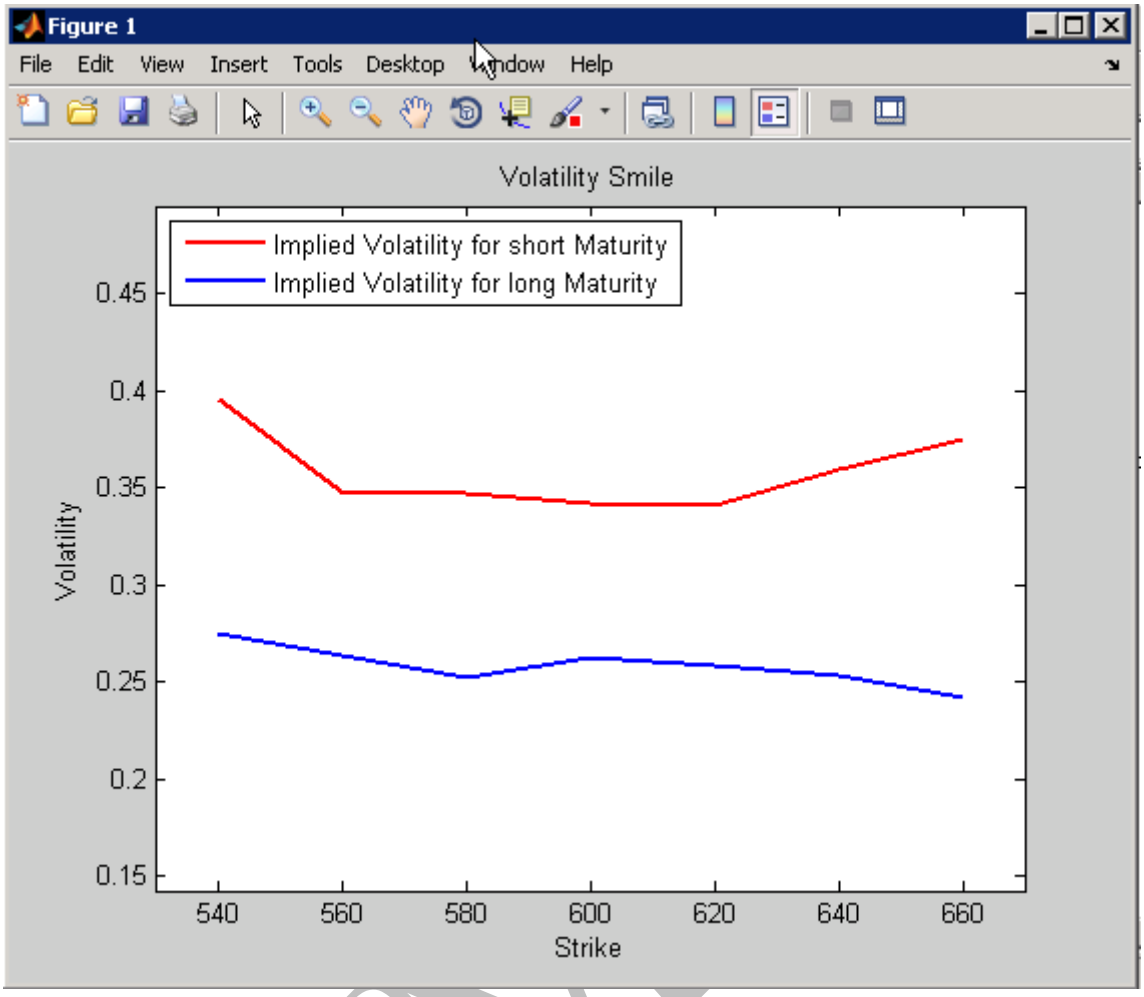
EDITOR PUBLISH VIEW

```
function [value] = diffbls2price(PutOrCall,S0,X,r,sigma,T,div,Optionprice)
    %computes the difference between the function
    %BlackScholes option value and the given option price.
    [call, put] = blsprice(S0,X,r,T,sigma,div);
    switch PutOrCall
        case 'Call'
            value = call - Optionprice;
        case 'Put'
            value = put - Optionprice;
        otherwise
            error('Check option type!');
    end
end
```

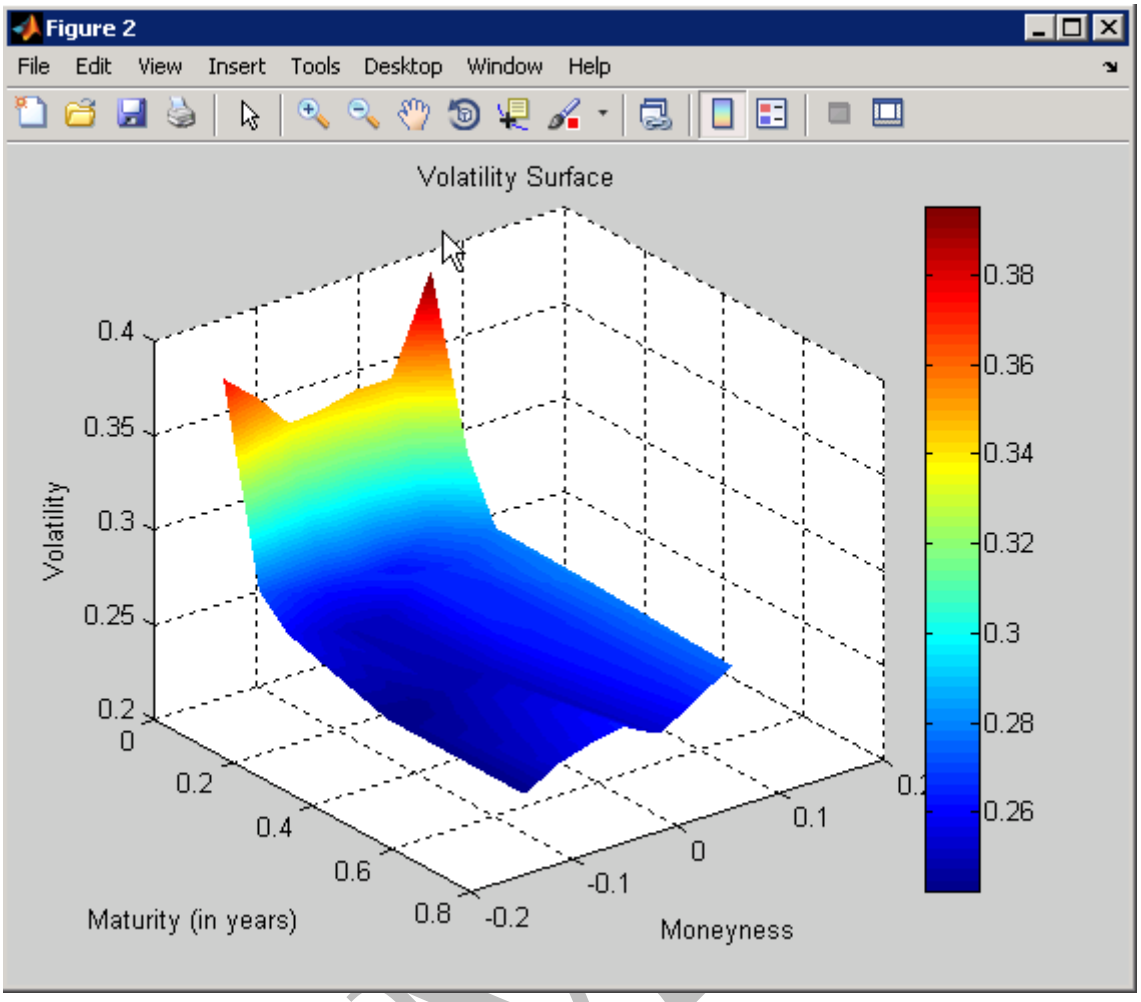
Command Window

```
implVolM =

    0.38    0.27    0.26    0.24    0.24
    0.36    0.27    0.25    0.24    0.25
    0.34    0.26    0.25    0.25    0.26
    0.34    0.27    0.25    0.25    0.26
    0.35    0.26    0.27    0.26    0.25
    0.35    0.28    0.27    0.27    0.26
    0.40    0.31    0.28    0.28    0.28
```


Figure 1
File Edit View Insert Tools Desktop Window Help
Volatility Smile
Implied Volatility for short Maturity
Implied Volatility for long Maturity
0.45
0.4
0.35
0.3
0.25
0.2
0.15
Volatility
540 560 580 600 620 640 660
Strike

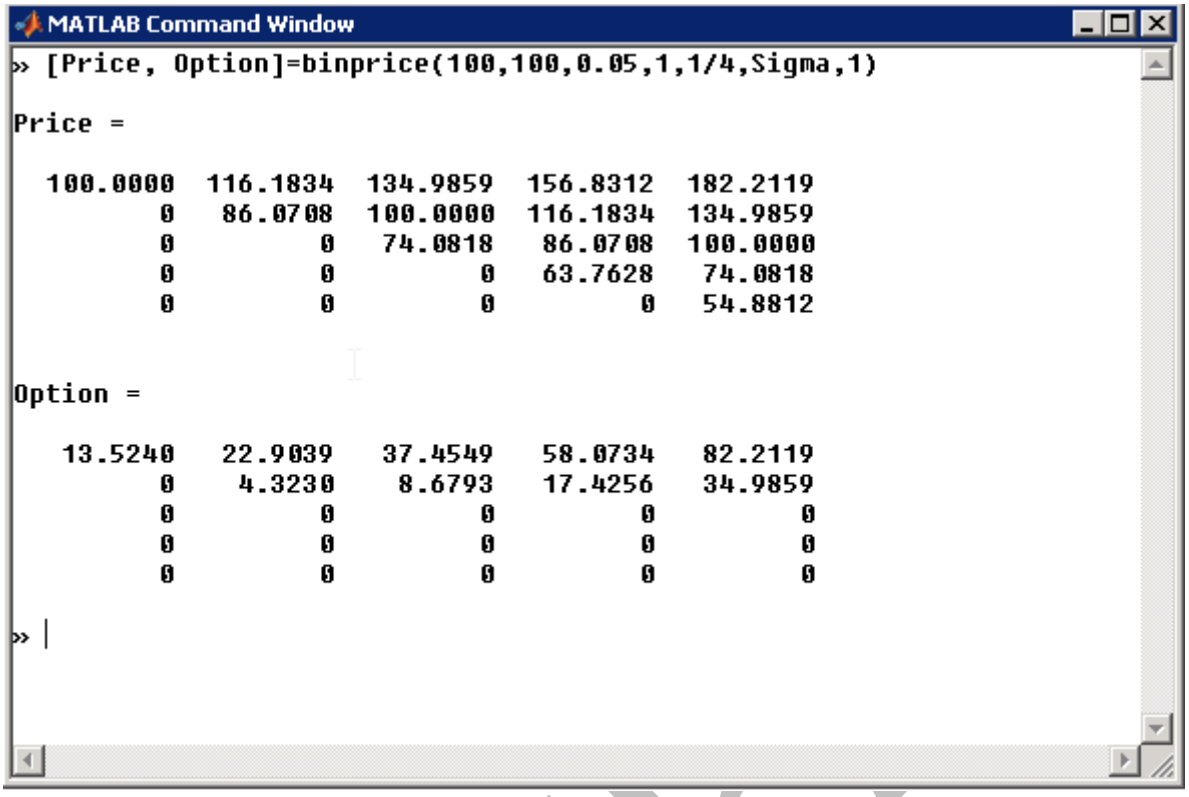
SAMPLE



CRR binomial European option pricing model

The last argument "1" means we want the price for a Call. Otherwise if we want the same for a Put we have to write "0".

SAMPLE


MATLAB Command Window
» [Price, Option]=binprice(100,100,0.05,1,1/4,Sigma,1)
Price =
100.0000 116.1834 134.9859 156.8312 182.2119
0 86.0708 100.0000 116.1834 134.9859
0 0 74.0818 86.0708 100.0000
0 0 0 63.7628 74.0818
0 0 0 0 54.8812
Option =
13.5240 22.9039 37.4549 58.0734 82.2119
0 4.3230 8.6793 17.4256 34.9859
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
»

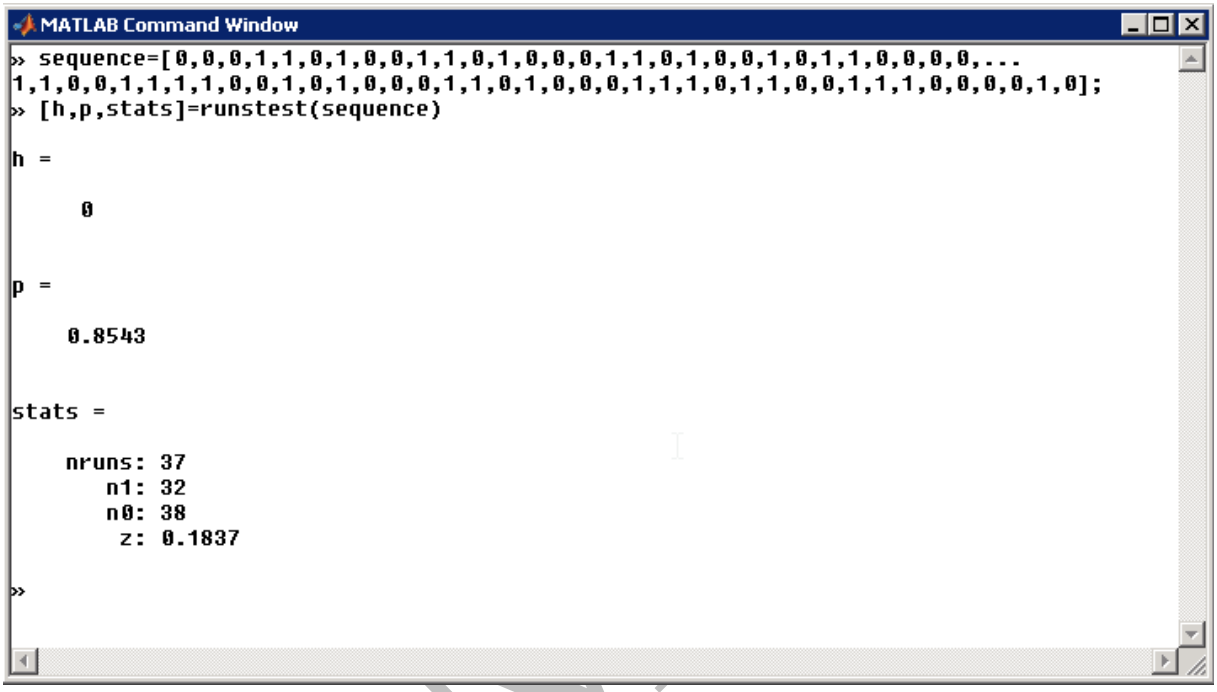
SAMPLE

Time series

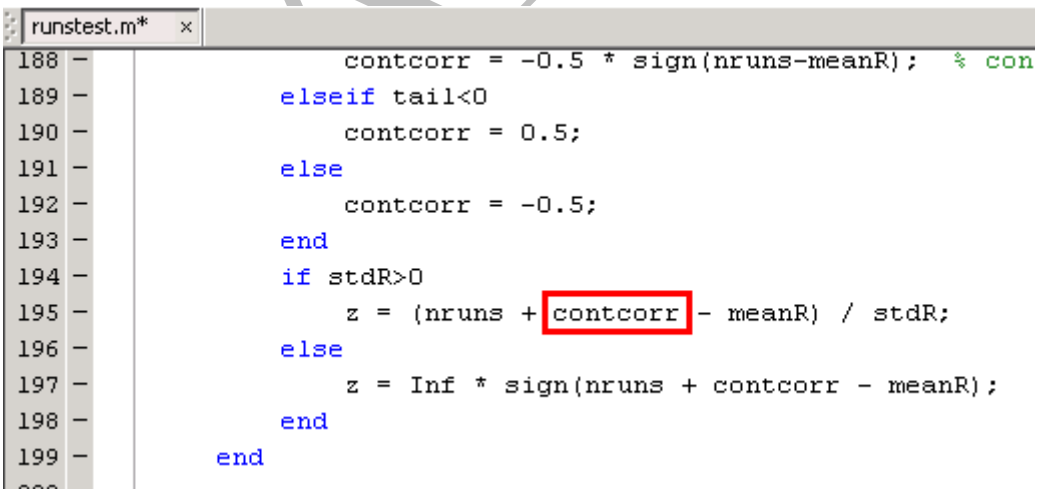
We will here focus only on ARIMA techniques!

Runs test (Wald-Wolfowitz)

Here the purpose is to do a typical test of randomness of Wald-Wolfowitz using the same values as in the theoretical course after we proved the method of the test:



If we don't get the same p-value as in the theoretical course, this is because of the continuity correction term in the Normal approximation of MATLAB:



We can the MATLAB script as following:

runstest.m

```
172 -       nruns = 1 + sum(x(1:end-1) ~= x(2:end));
173
174         % Normal statistic for asymptotic test
175 -       if N==1
176 -           z = NaN;
177 -       else
178 -           if updown
179 -               meanR = (2*N-1)/3;
180 -               stdR = sqrt((16*N-29)/90);
181 -           else
182 -               meanR = 1 + 2*n1*n0/N;
183 -               stdR = sqrt(2*n1*n0*(2*n1*n0-N) / (N^2 * (N-1)));
184 -               meanR
185 -               stdR^2
186 -           end
```

So we can see we get the same mean and variance as calculated by hand in the theoretical course:

MATLAB Command Window

```
>> [h,p,stats]=runstest(sequence)

meanR =

   35.7429


ans =

   16.9902


h =

     0


p =

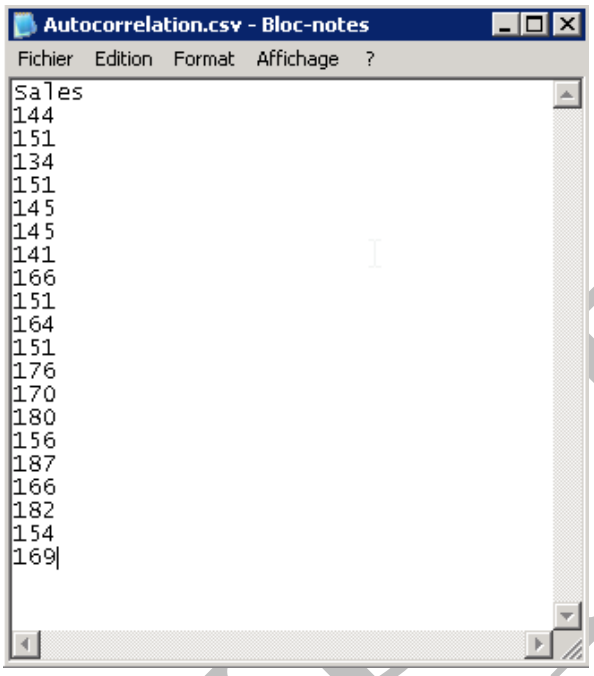
    0.8543


stats =

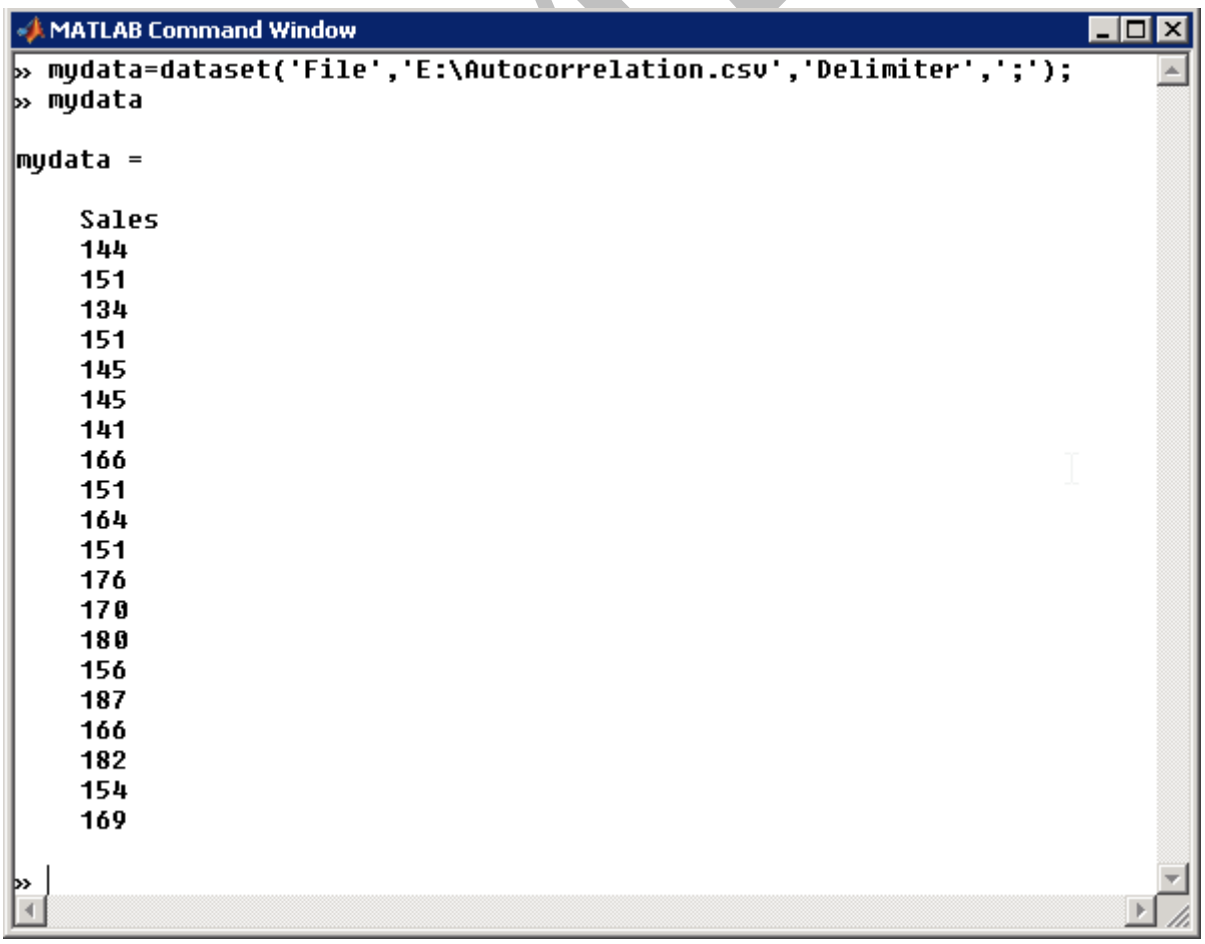
    nruns: 37
       n1: 32
       n0: 38
        z: 0.1837
```

Autocorrelation

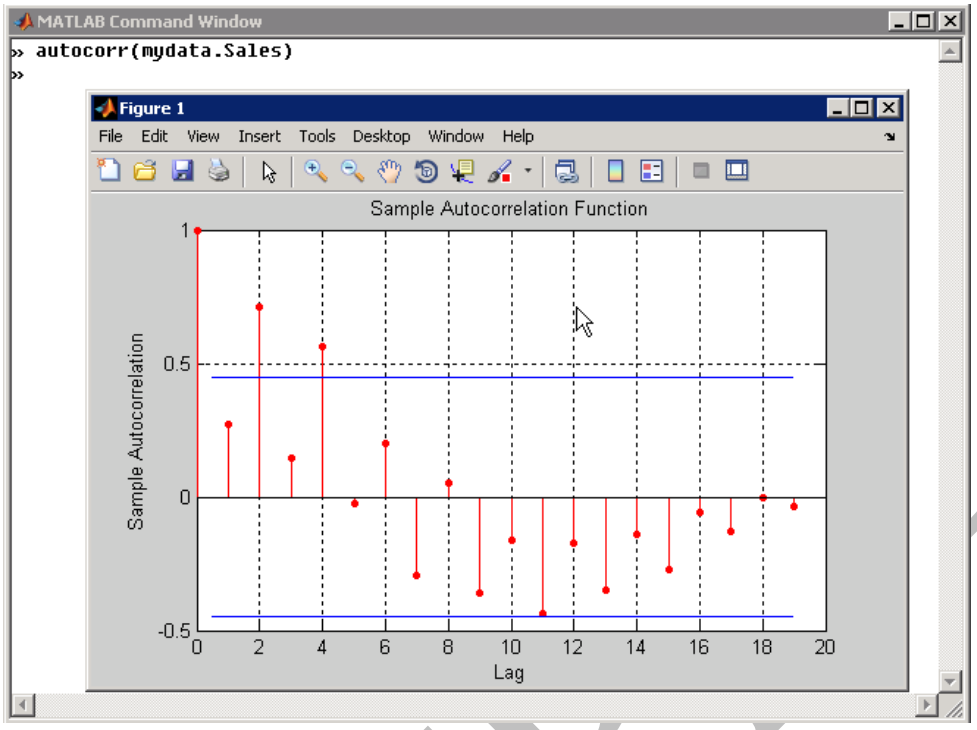
We will use the same data set as in the theoretical course:



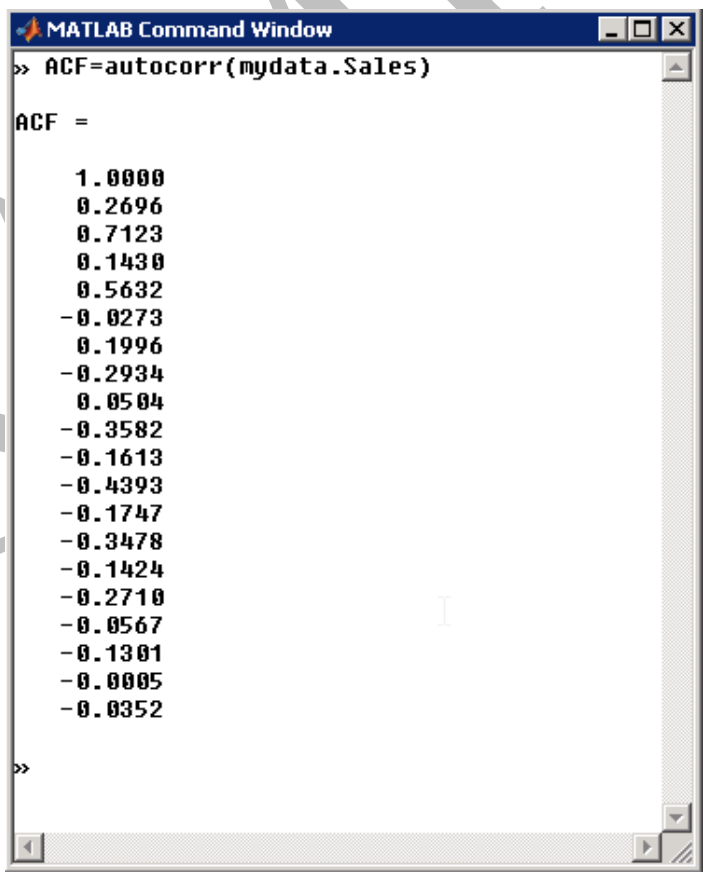
We load the data as normally:



Now we use the function **autocorr( ):**



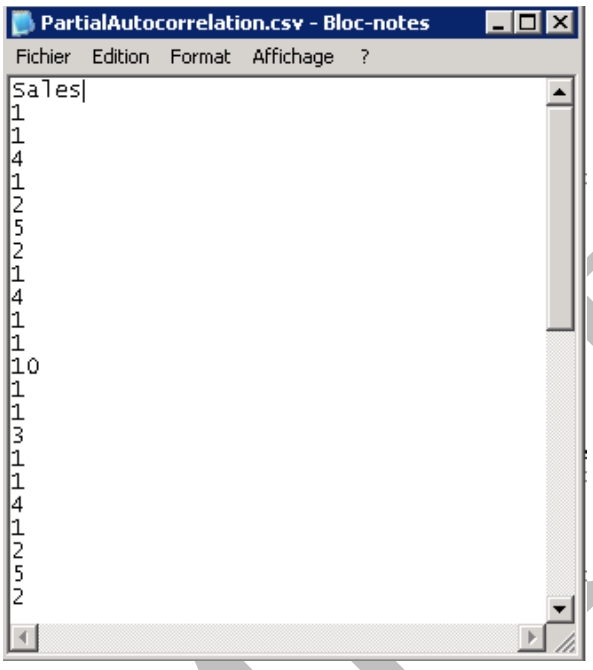
But with the values this is much better:



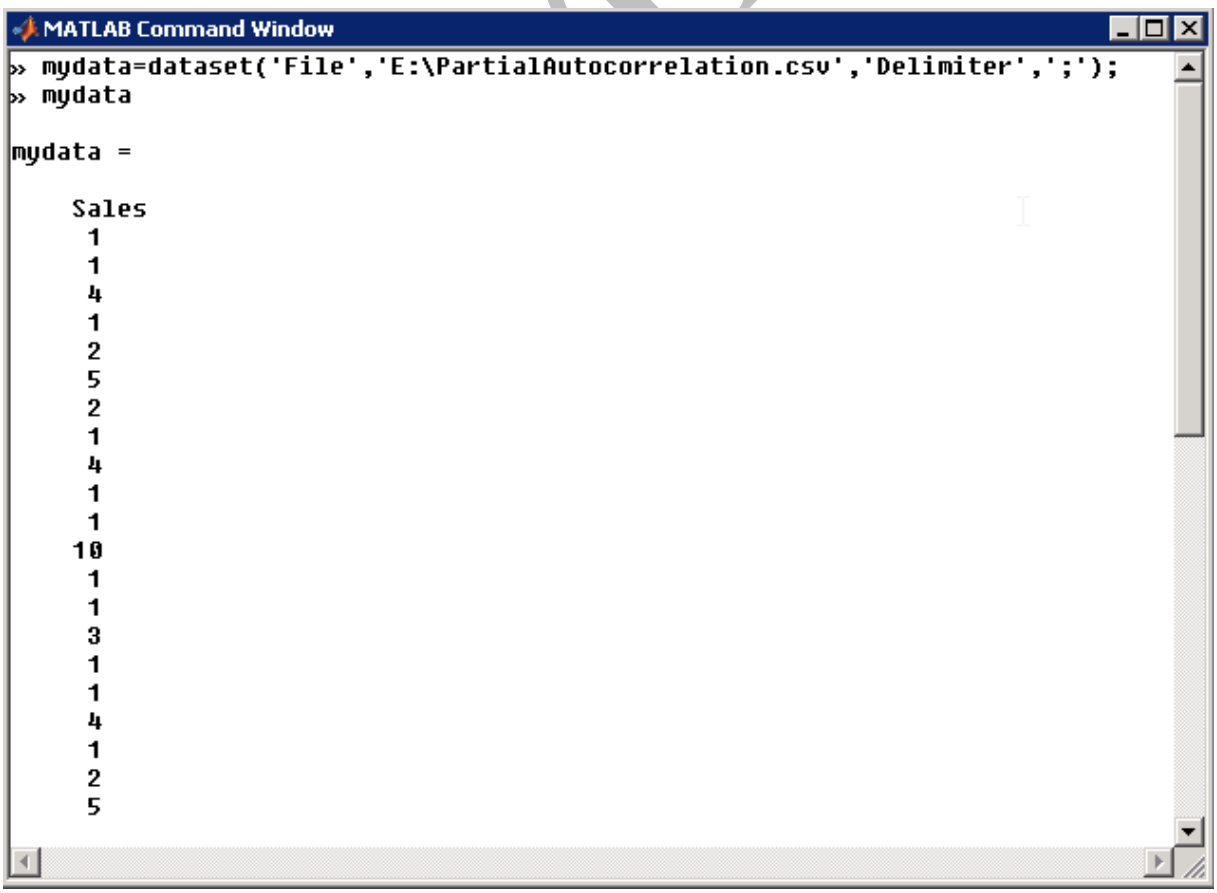
And as we can see the values are almost perfectly matching.

Partial autocorrelation

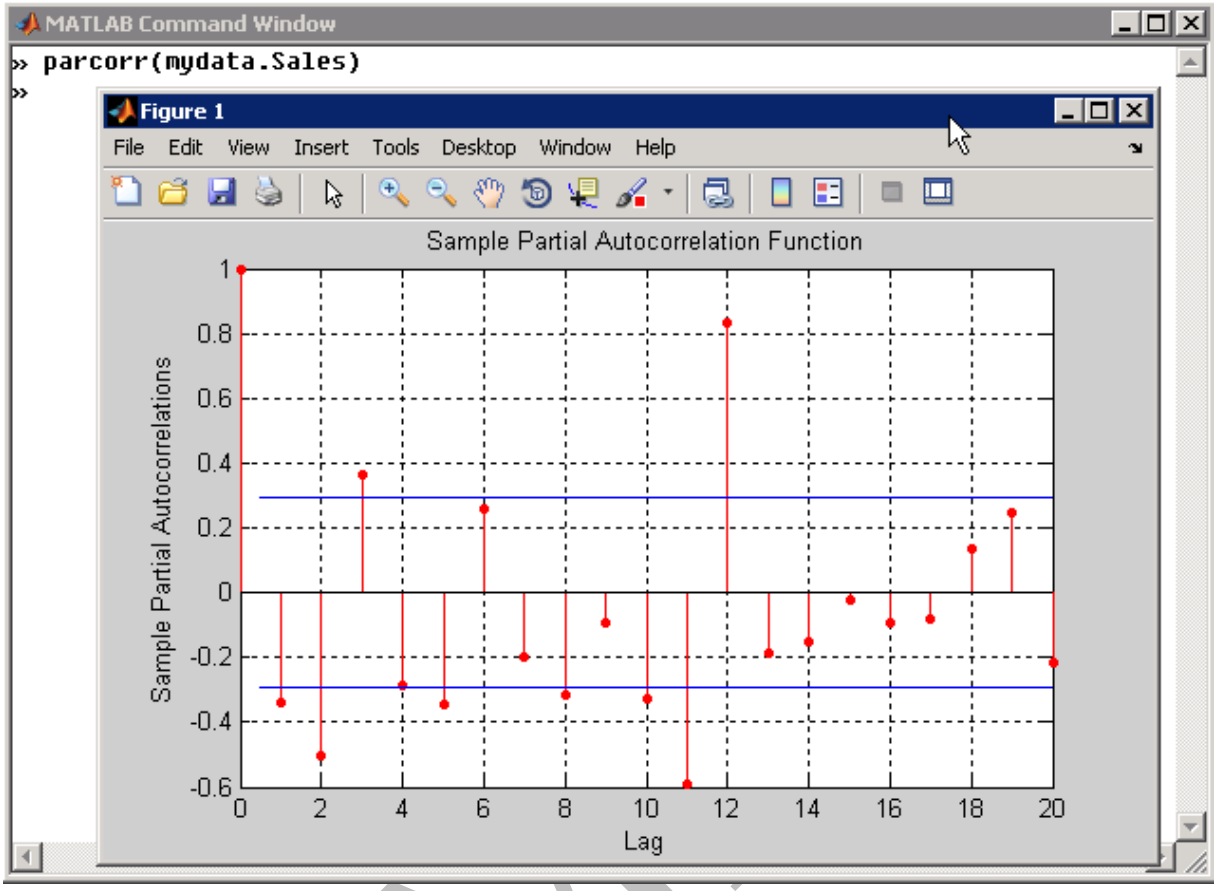
We still stay with the same data set as in the theoretical course. If you remember doing the calculations by hand was very very boring. So first we have:



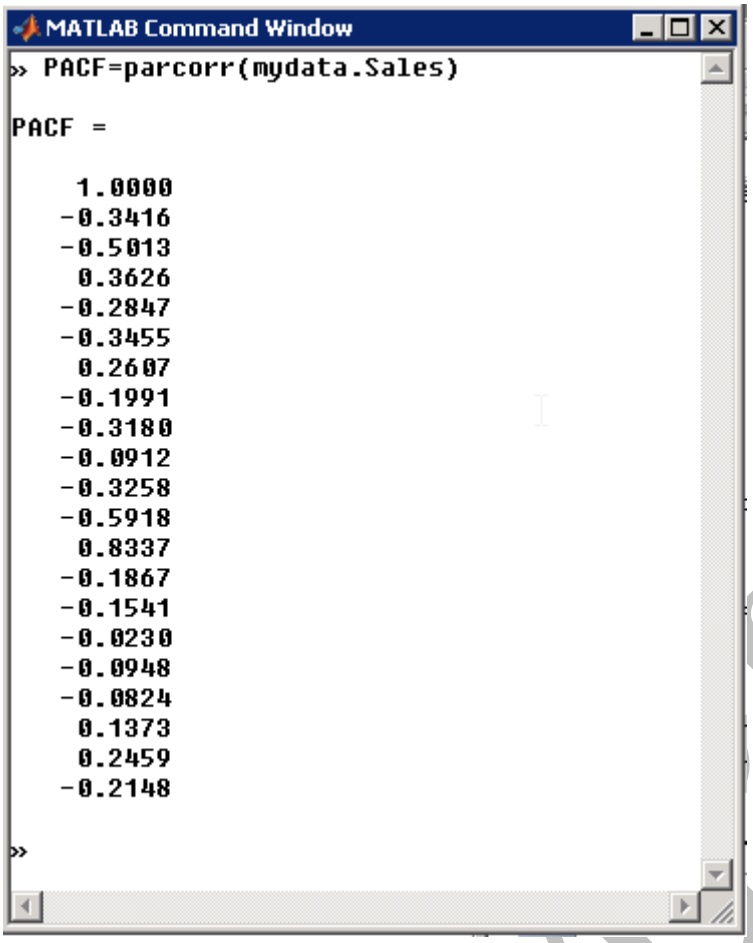
We load the dataset:



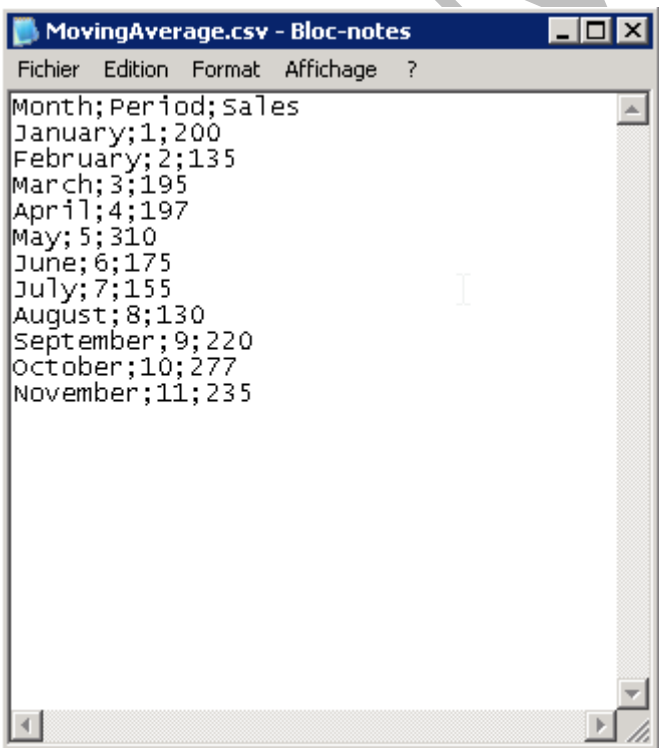
And we get:

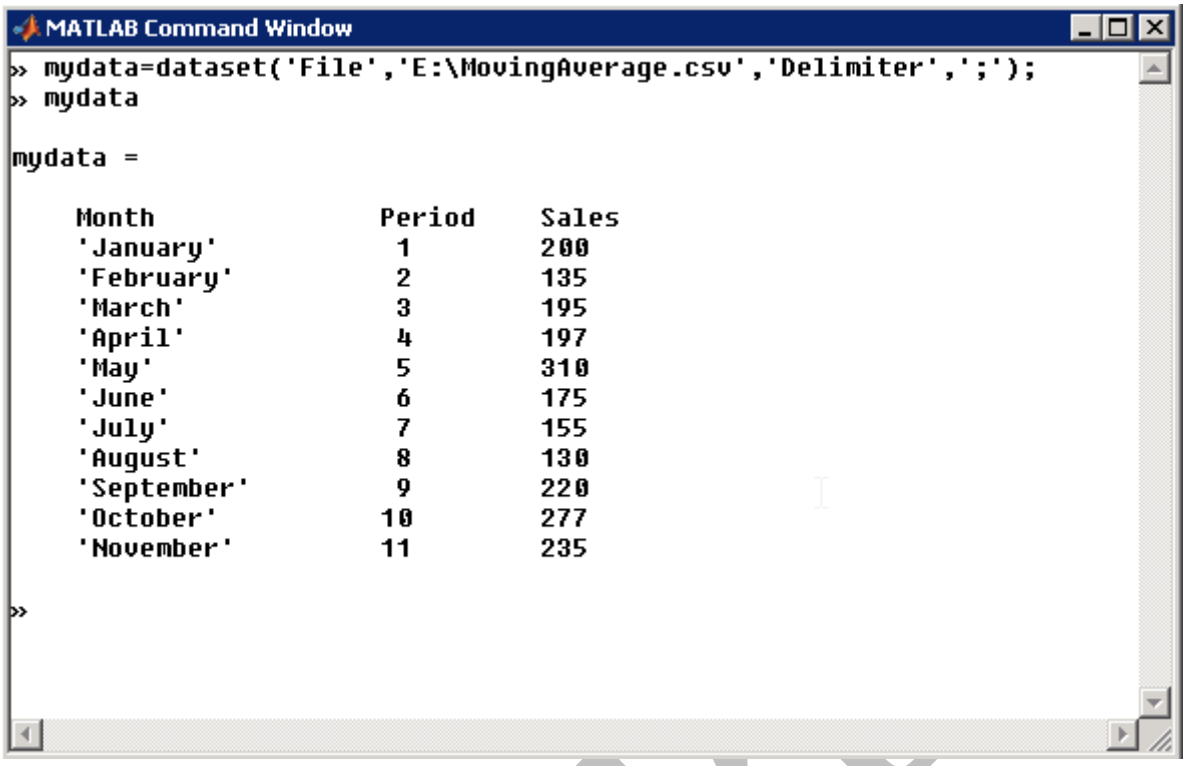


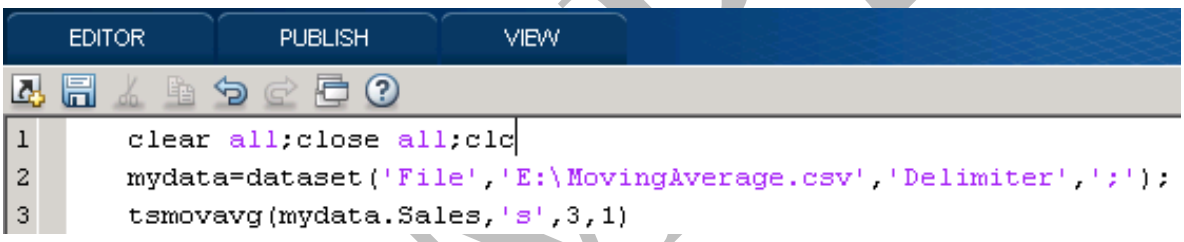
SAMPLE

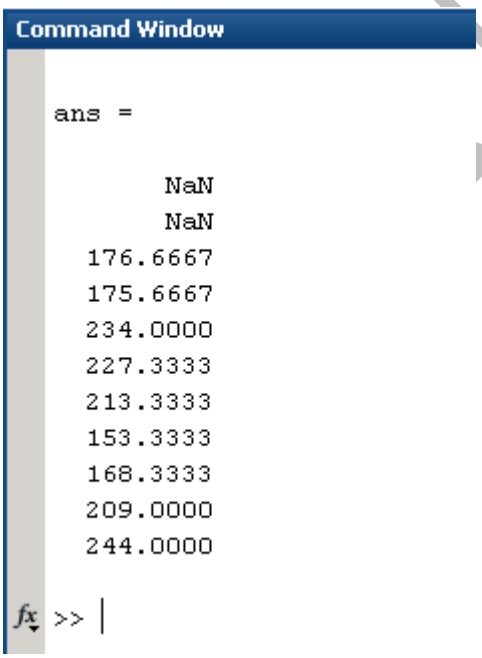


Moving average MM3 (without ARIMA)

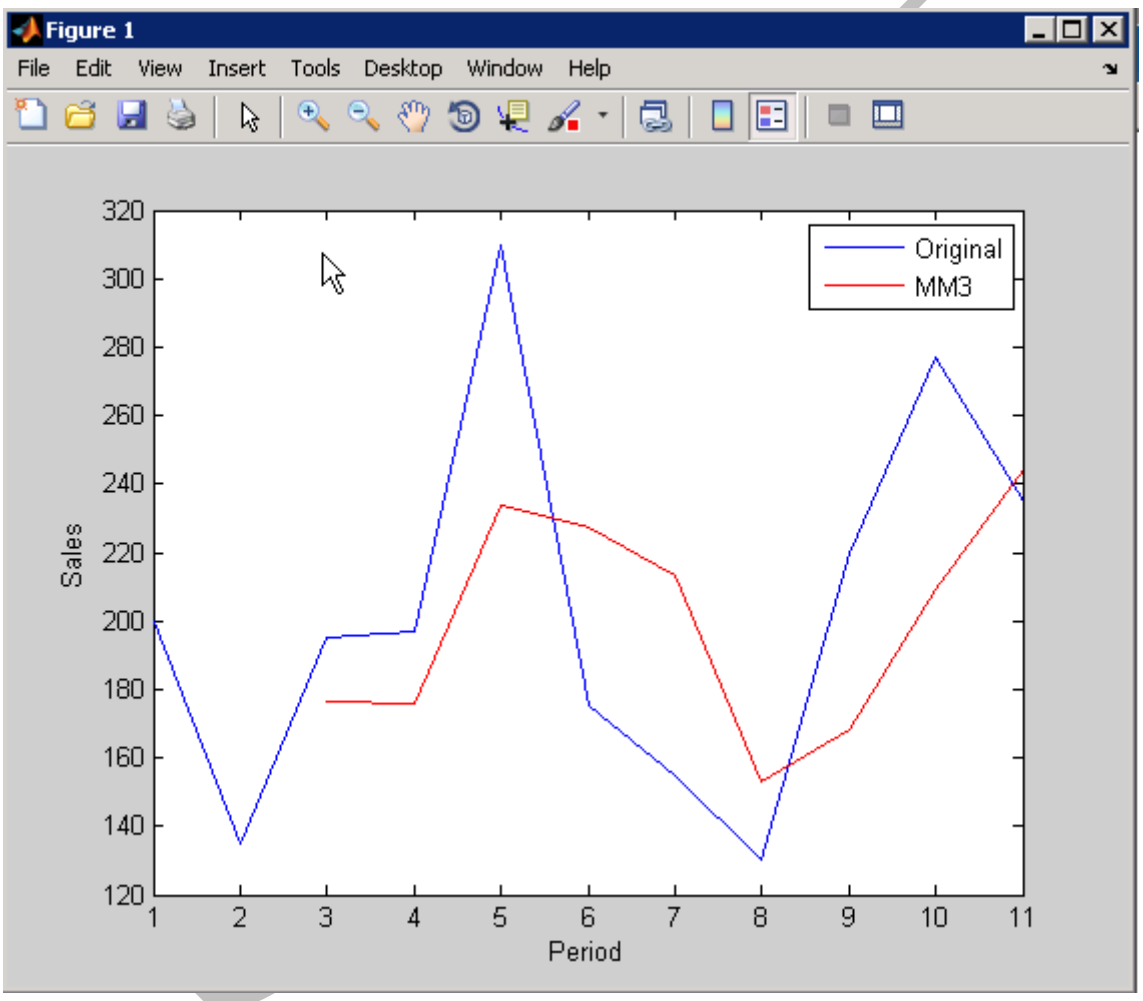



MATLAB Command Window
» mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
» mydata
mydata =
Month Period Sales
'January' 1 200
'February' 2 135
'March' 3 195
'April' 4 197
'May' 5 310
'June' 6 175
'July' 7 155
'August' 8 130
'September' 9 220
'October' 10 277
'November' 11 235
»


EDITOR
PUBLISH
VIEW
1 clear all;close all;clc
2 mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
3 tsmovavg(mydata.Sales,'s',3,1)


Command Window
ans =
NaN
NaN
176.6667
175.6667
234.0000
227.3333
213.3333
153.3333
168.3333
209.0000
244.0000
>>

```
clear all;close all;clc
mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
tsmovavg(mydata.Sales,'s',3,1)
plot(mydata.Period,mydata.Sales,'b');
hold on;
plot(mydata.Period,tsmovavg(mydata.Sales,'s',3,1),'r');
xlabel('Period');ylabel('Sales');
legend('Original','MM3');
```



Simple Exponential Weighted Moving Average (EWMA)

```
clear all;close all;clc
Y=[143 152 161 139 137 174 142 141 162 180 164 171 206 193 207 218 229 225 204 227 223 242 239 266];
tsmovavg(Y,'e',2)'
```

Command Window

```
ans =

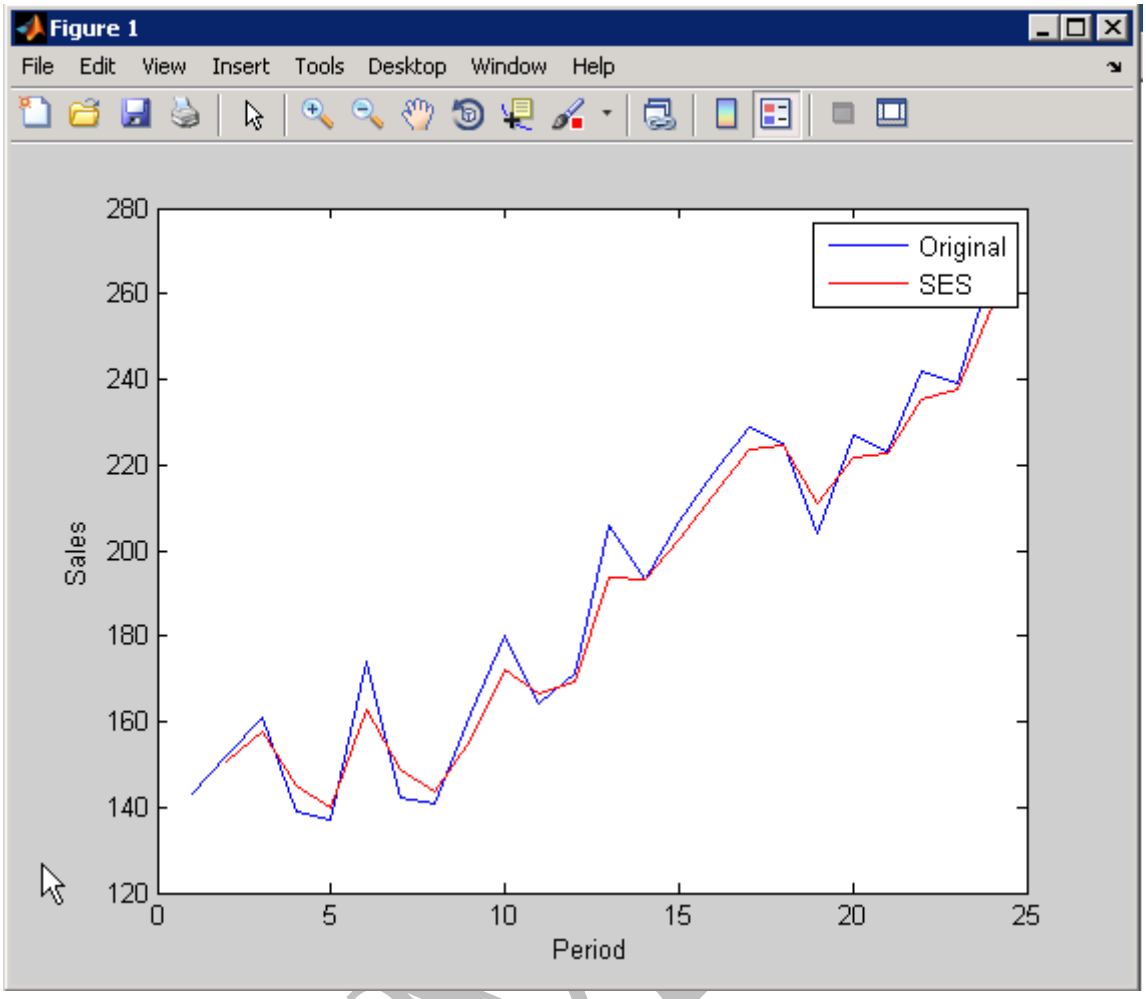
       NaN
  150.5000
  157.5000
  145.1667
  139.7222
  162.5741
  148.8580
  143.6193
  155.8731
  171.9577
  166.6526
  169.5509
  193.8503
  193.2834
  202.4278
  212.8093
  223.6031
  224.5344
  210.8448
  221.6149
  222.5383
  235.5128
  237.8376
  256.6125

>>
```

EDITOR PUBLISH VIEW

```
1  clear all;close all;clc
2  Y=[143 152 161 139 137 174 142 141 162 180 164 171 206 193 207 218 229 225 204 227 223 242 239 266];
3  tsmovavg(Y,'e',2)'
4  plot(1:length(Y),Y,'b');
5  hold on;
6  plot(1:length(Y),tsmovavg(Y,'e',2),'r');
7  xlabel('Period');ylabel('Sales');
8  legend('Original','SES');
```

EXAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
SES
Sales
Period
280
260
240
220
200
180
160
140
120
0
5
10
15
20
25

SAM

Univariate AR(2)/ARIMA(2,0,0)

EDITOR PUBLISH VIEW

```
clear all;clc
arima(2,0,0)
Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
Mdl=arima(2,0,0);
EstMdl=estimate(Mdl,Y')
ForecastPeriods=10;
[YF YMSE]=forecast(EstMdl,ForecastPeriods,'Y0',Y');

plot(Y,'Color',[.7,.7,.7]);
hold on
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
title('30-Period Forecasts and Approximate 95% Confidence Intervals');

%this part is to obtain values for backtesting
residuals=infer(EstMdl,Y');
BackTesting=Y'-residuals
```

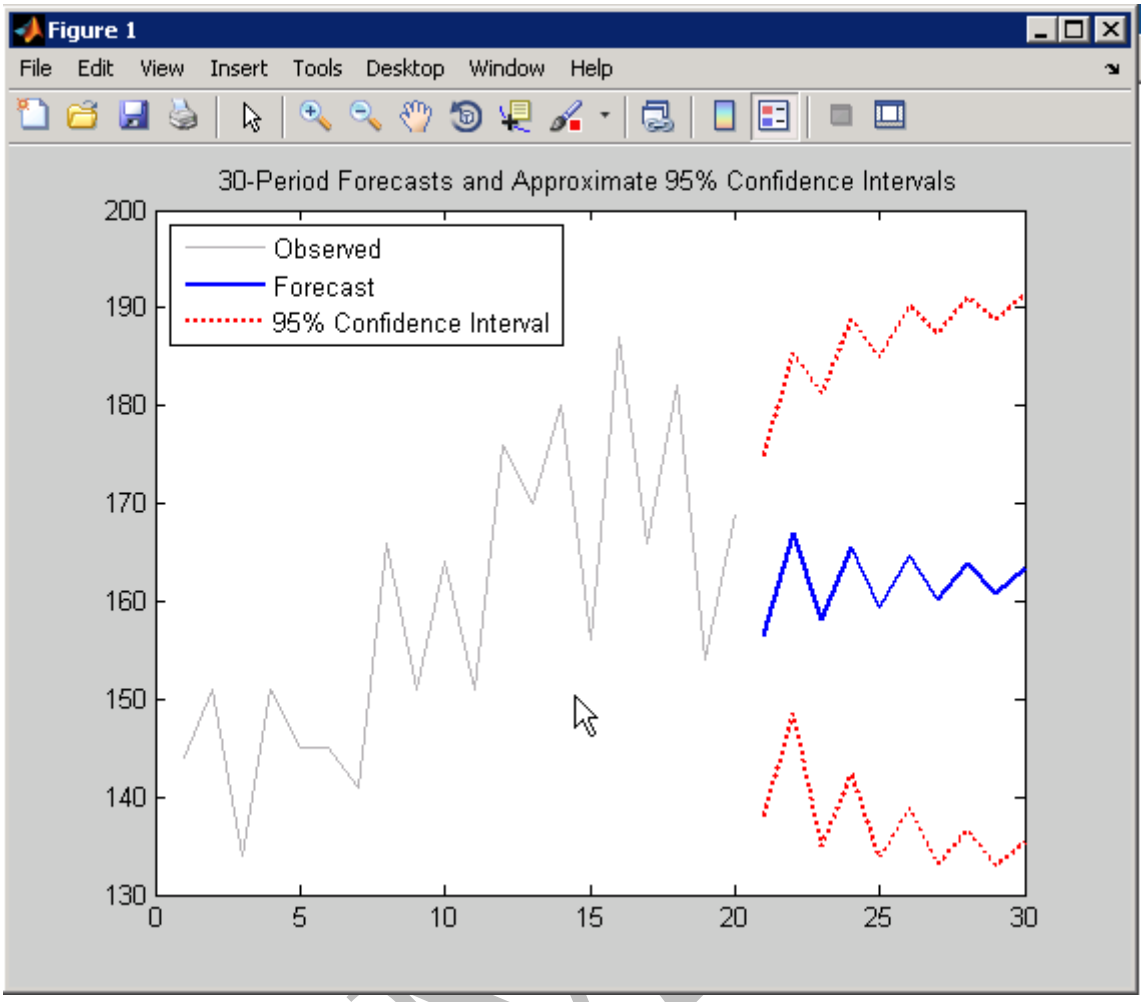
Command Window

```
    ARIMA(2,0,0) Model:
    --------------------
    Conditional Probability Distribution: Gaussian

                                 Standard          t
     Parameter       Value          Error       Statistic
    -----------   -----------   ------------   -----------
     Constant       27.8249        30.3068       0.918107
        AR{1}     0.0680209        0.16081       0.422988
        AR{2}      0.760531       0.175242        4.33989
     Variance       88.0789        45.2865        1.94493

EstMdl =

    ARIMA(2,0,0) Model:
    --------------------
    Distribution: Name = 'Gaussian'
               P: 2
               D: 0
               Q: 0
        Constant: 27.8249
              AR: {0.0680209 0.760531} at Lags [1 2]
             SAR: {}
              MA: {}
             SMA: {}
        Variance: 88.0789
```


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
200
190
180
170
160
150
140
130
0
5
10
15
20
25
30

SAM

```
Command Window
  BackTesting=Y'-residuals

  BackTesting =

    150.5348
    153.5706
    147.6125
    151.7799
    140.0072
    152.5281
    147.9649
    147.6929
    146.3513
    164.3442
    153.8205
    162.8232
    154.6368
    173.2419
    169.3589
    175.3318
    159.1877
    181.3357
    166.4529
    176.7168
```

SAMPLE

Univariate ARIMA(0,0,0)/Moving Average

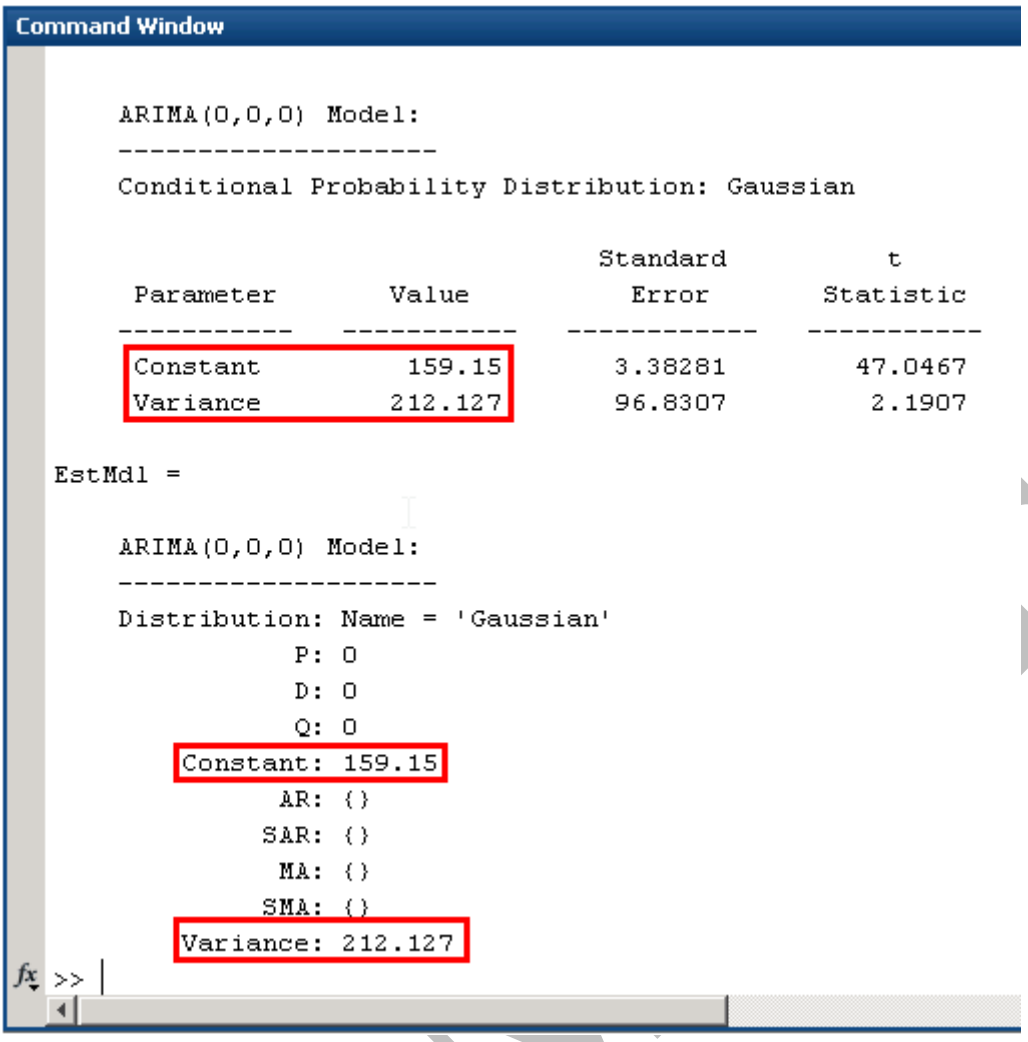
EDITOR PUBLISH VIEW

```
clear all;clc
arima(2,0,0)
Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
Mdl=arima(0,0,0);
EstMdl=estimate(Mdl,Y')
ForecastPeriods=10;
[YF YMSE]=forecast(EstMdl,ForecastPeriods,'Y0',Y');

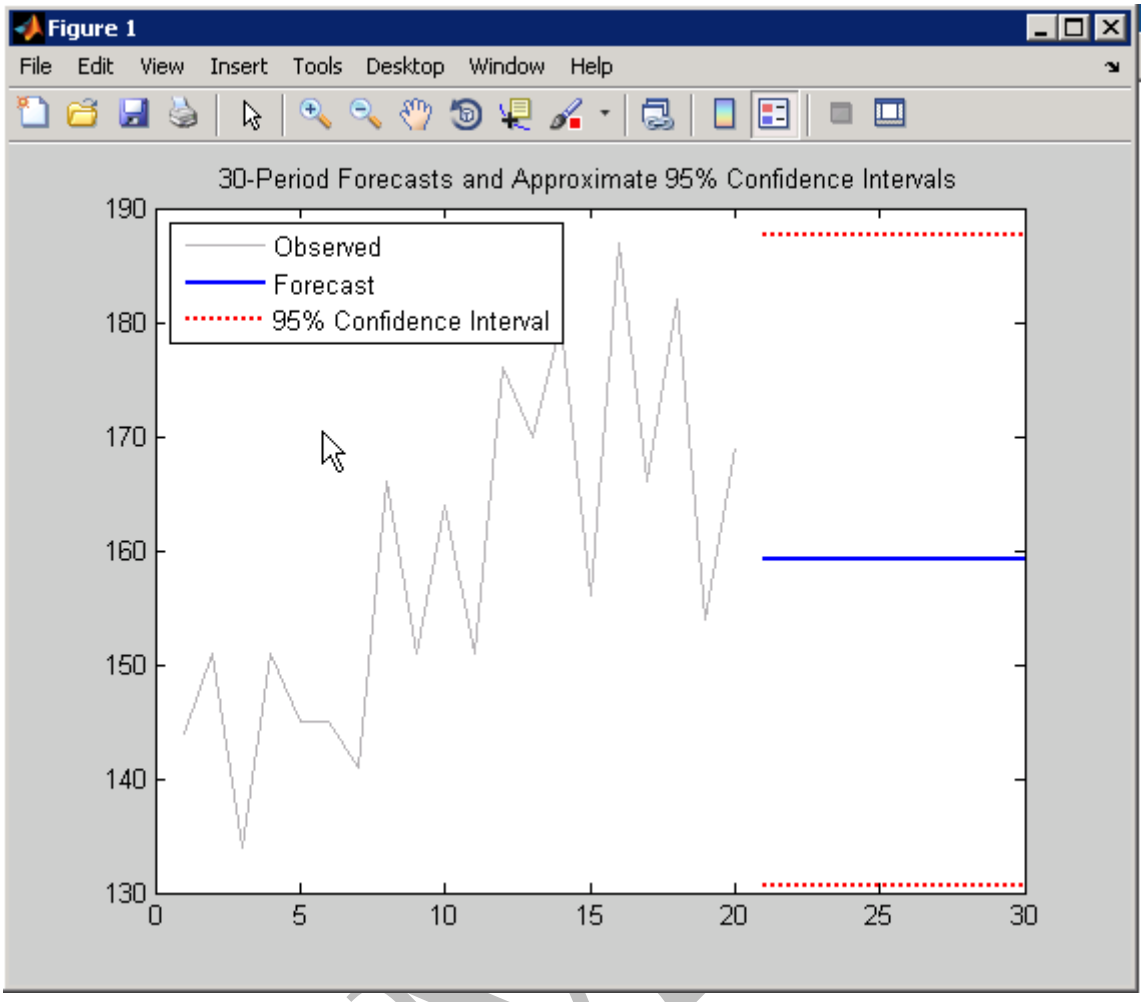
plot(Y,'Color',[.7,.7,.7]);
hold on
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
title('30-Period Forecasts and Approximate 95% Confidence Intervals');

%this part is to obtain values for backtesting
residuals=infer(EstMdl,Y');
BackTesting=Y'-residuals
```

SAMPLE


Command Window
ARIMA(0,0,0) Model:
Conditional Probability Distribution: Gaussian
Parameter Value Standard Error t Statistic
Constant 159.15 3.38281 47.0467
Variance 212.127 96.8307 2.1907
EstMdl =
ARIMA(0,0,0) Model:
Distribution: Name = 'Gaussian'
P: 0
D: 0
Q: 0
Constant: 159.15
AR: {}
SAR: {}
MA: {}
SMA: {}
Variance: 212.127
>>

SAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
190
180
170
160
150
140
130
0
5
10
15
20
25
30

SAM

```
Command Window

BackTesting =

  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499
  159.1499

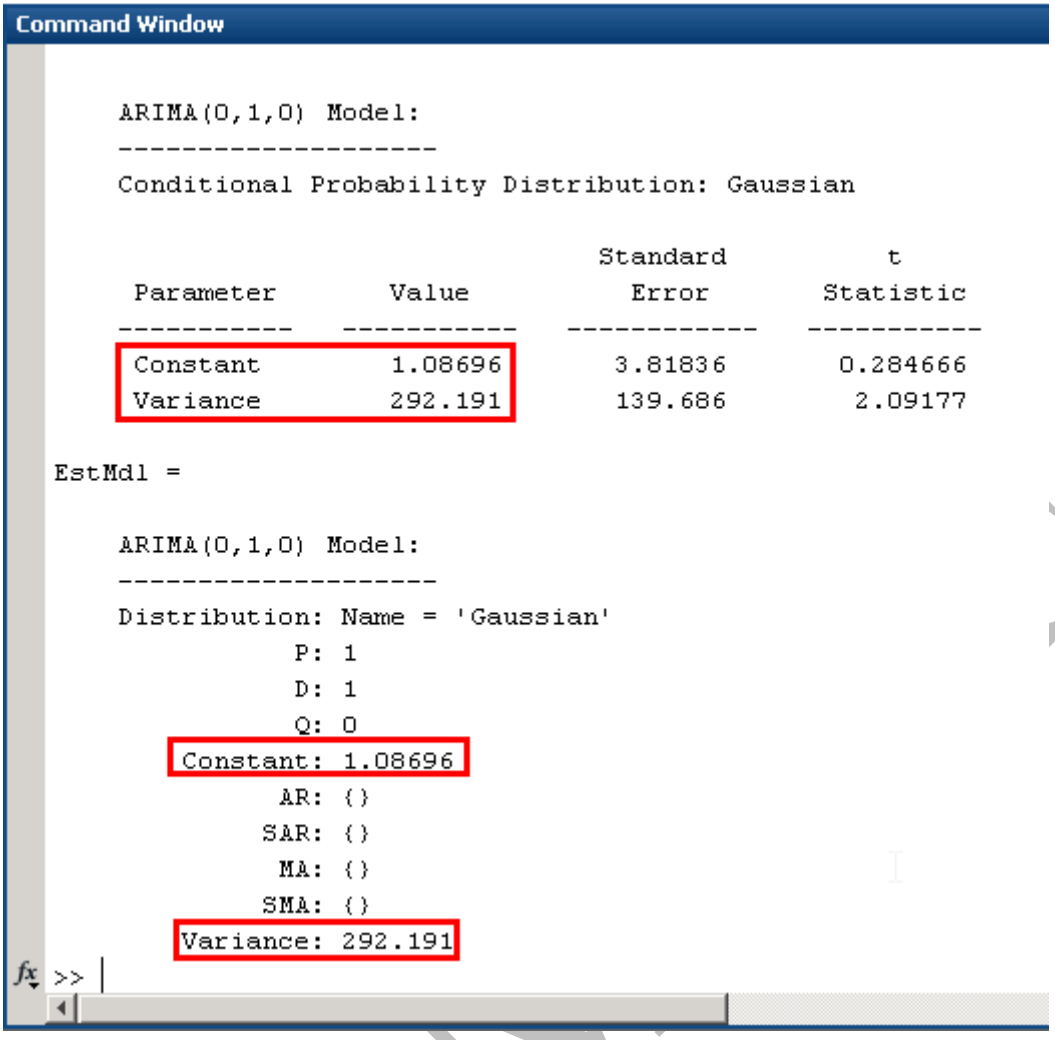
>>
```

Univariate ARIMA(0,1,0)

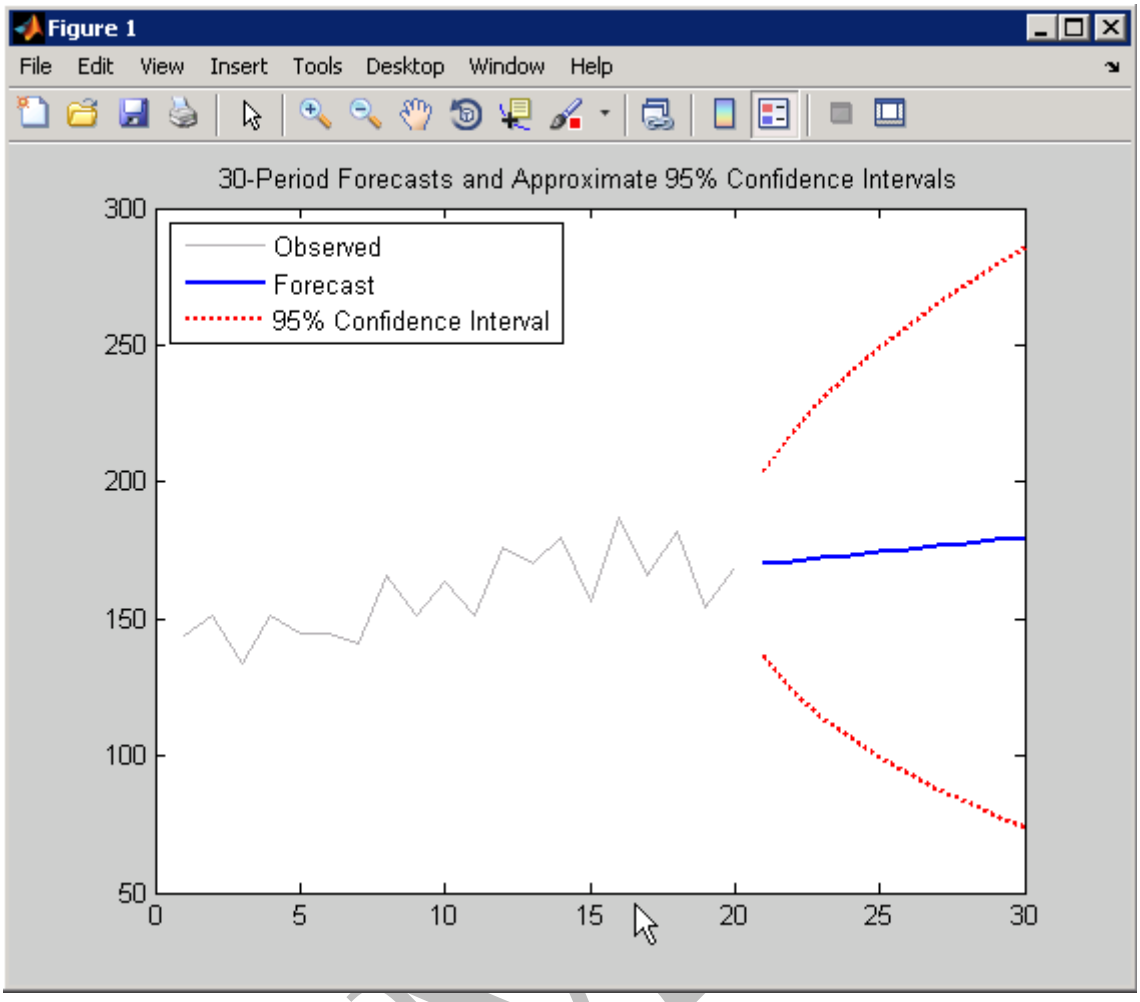
```
EDITOR  PUBLISH  VIEW

1  clear all;clc
2  arima(2,0,0)
3  Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4  Mdl=arima(0,1,0);
5  EstMdl=estimate(Mdl,Y')
6  ForecastPeriods=10;
7  [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9  plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals
```

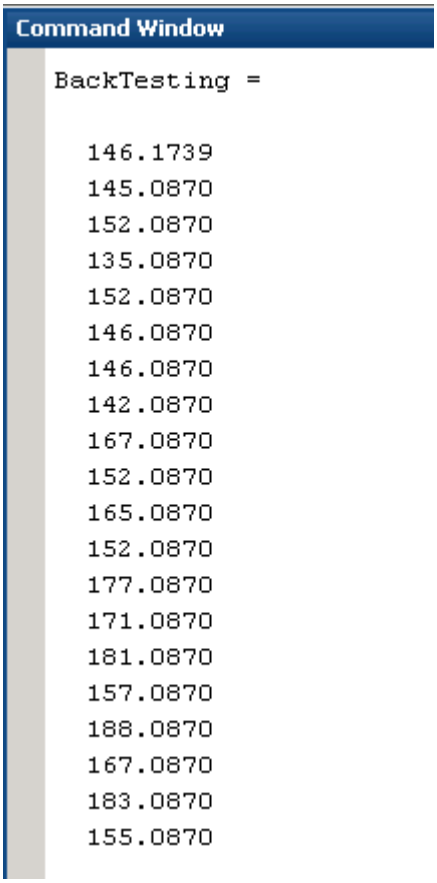
MPLE


Command Window
ARIMA(0,1,0) Model:
Conditional Probability Distribution: Gaussian
Parameter Value Standard Error t Statistic
Constant 1.08696 3.81836 0.284666
Variance 292.191 139.686 2.09177
EstMdl =
ARIMA(0,1,0) Model:
Distribution: Name = 'Gaussian'
P: 1
D: 1
Q: 0
Constant: 1.08696
AR: {}
SAR: {}
MA: {}
SMA: {}
Variance: 292.191
>>

SAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
300
250
200
150
100
50
0
5
10
15
20
25
30

SAM


Command Window
BackTesting =
146.1739
145.0870
152.0870
135.0870
152.0870
146.0870
146.0870
142.0870
167.0870
152.0870
165.0870
152.0870
177.0870
171.0870
181.0870
157.0870
188.0870
167.0870
183.0870
155.0870

SAMPLE

Univariate ARIMA(0,1,1)/Simple Exponential Smoothing (EWMA)

EDITOR PUBLISH VIEW

```
clear all;close all;clc
arima(2,0,0)
Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
Mdl=arima(0,1,1);
EstMdl=estimate(Mdl,Y')
ForecastPeriods=10;
[YF YMSE]=forecast(EstMdl,ForecastPeriods,'Y0',Y');

plot(Y,'Color',[.7,.7,.7]);
hold on
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
title('30-Period Forecasts and Approximate 95% Confidence Intervals');

%this part is to obtain values for backtesting
residuals=infer(EstMdl,Y');
BackTesting=Y'-residuals
```

Command Window

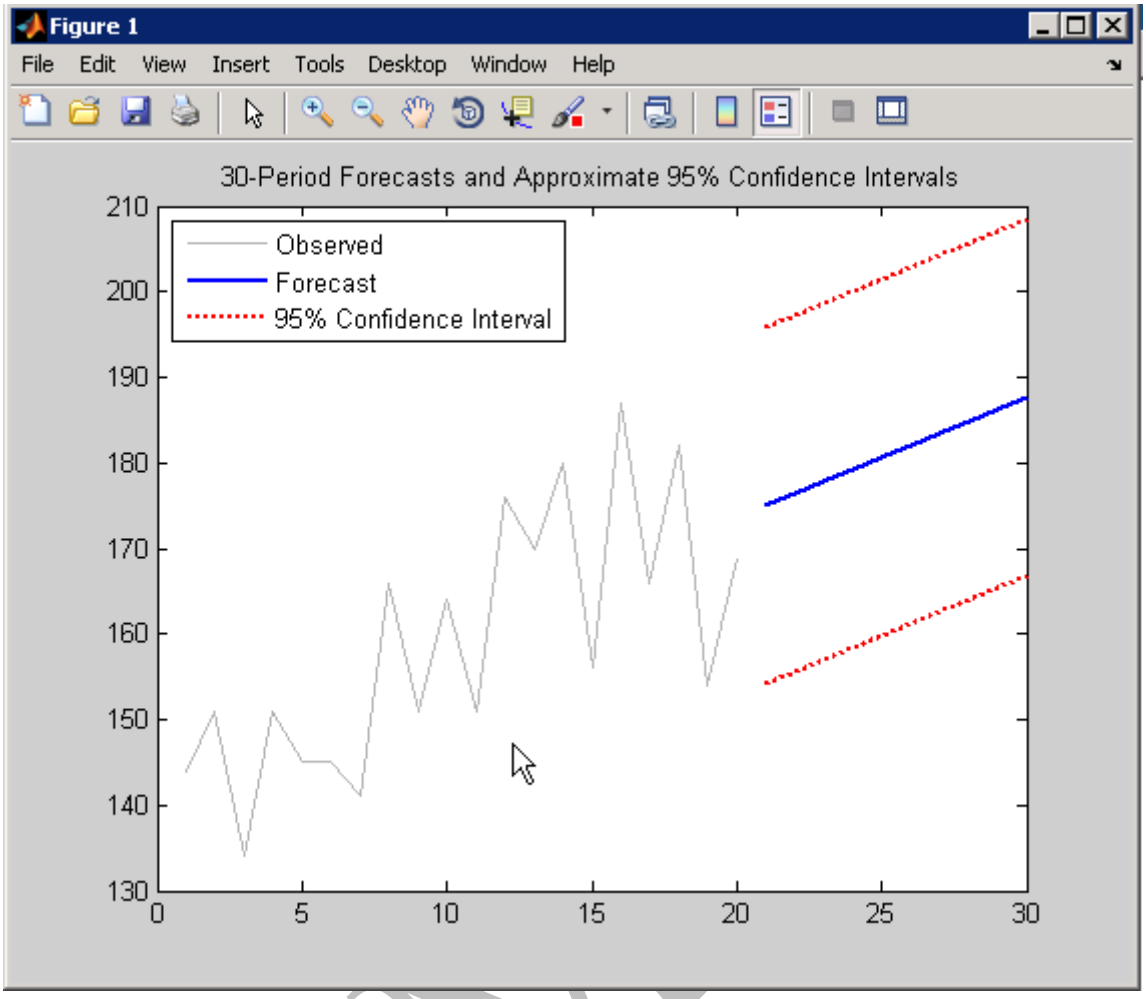
```
    ARIMA(0,1,1) Model:
    --------------------
    Conditional Probability Distribution: Gaussian

                                  Standard          t
     Parameter       Value          Error       Statistic
    -----------   -----------   ------------   -----------
     Constant        1.40876       0.23519        5.98988
        MA{1}             -1      0.128317       -7.79322
     Variance        113.197         56.493        2.00374

EstMdl =

    ARIMA(0,1,1) Model:
    --------------------
    Distribution: Name = 'Gaussian'
               P: 1
               D: 1
               Q: 1
        Constant: 1.40876
              AR: {}
             SAR: {}
              MA: {-1} at Lags [1]
             SMA: {}
        Variance: 113.197

fx >>
```


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
210
200
190
180
170
160
150
140
130
0
5
10
15
20
25
30

SAM

```
Command Window
BackTesting =

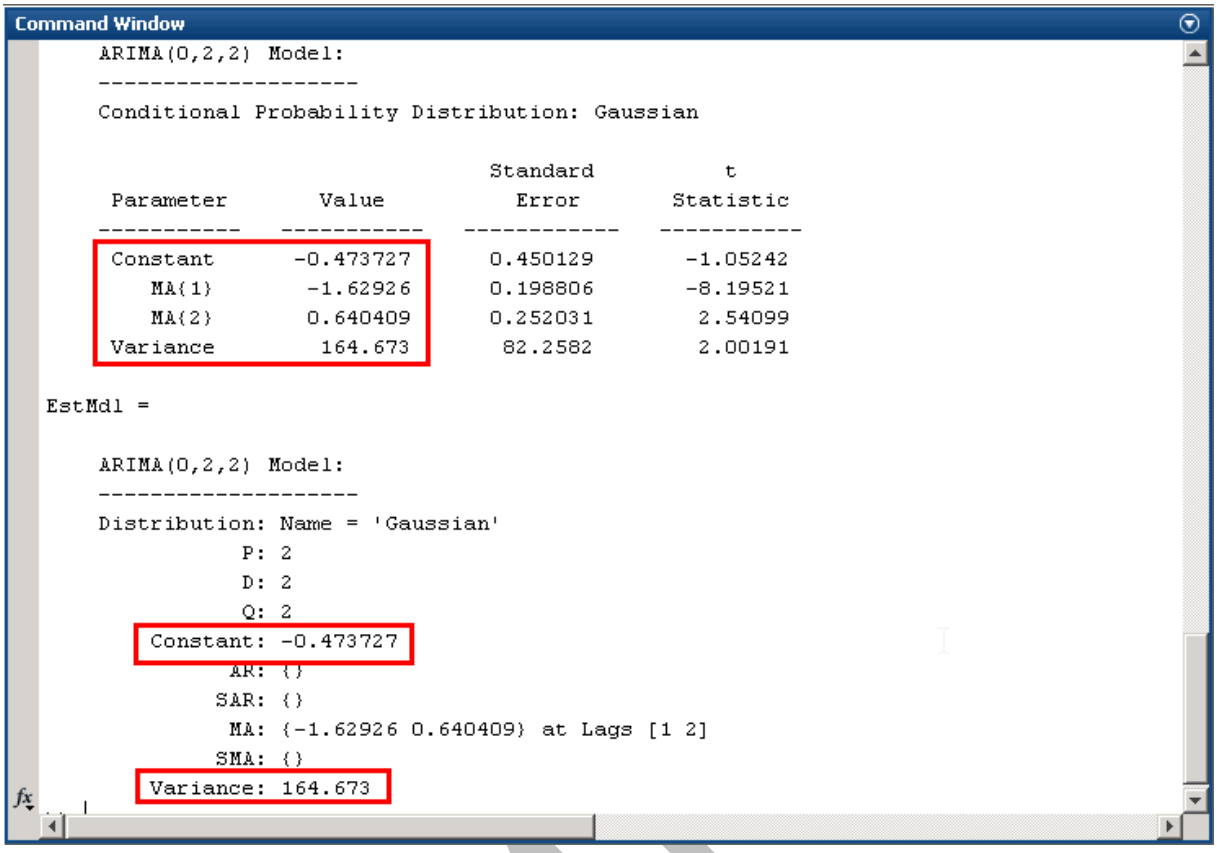
  146.8175
  148.2263
  149.6350
  151.0438
  152.4526
  153.8613
  155.2701
  156.6788
  158.0876
  159.4964
  160.9051
  162.3139
  163.7227
  165.1314
  166.5402
  167.9489
  169.3577
  170.7665
  172.1752
  173.5840
```

Univariate ARIMA(0,2,2)/Holt's double exponential smoothing

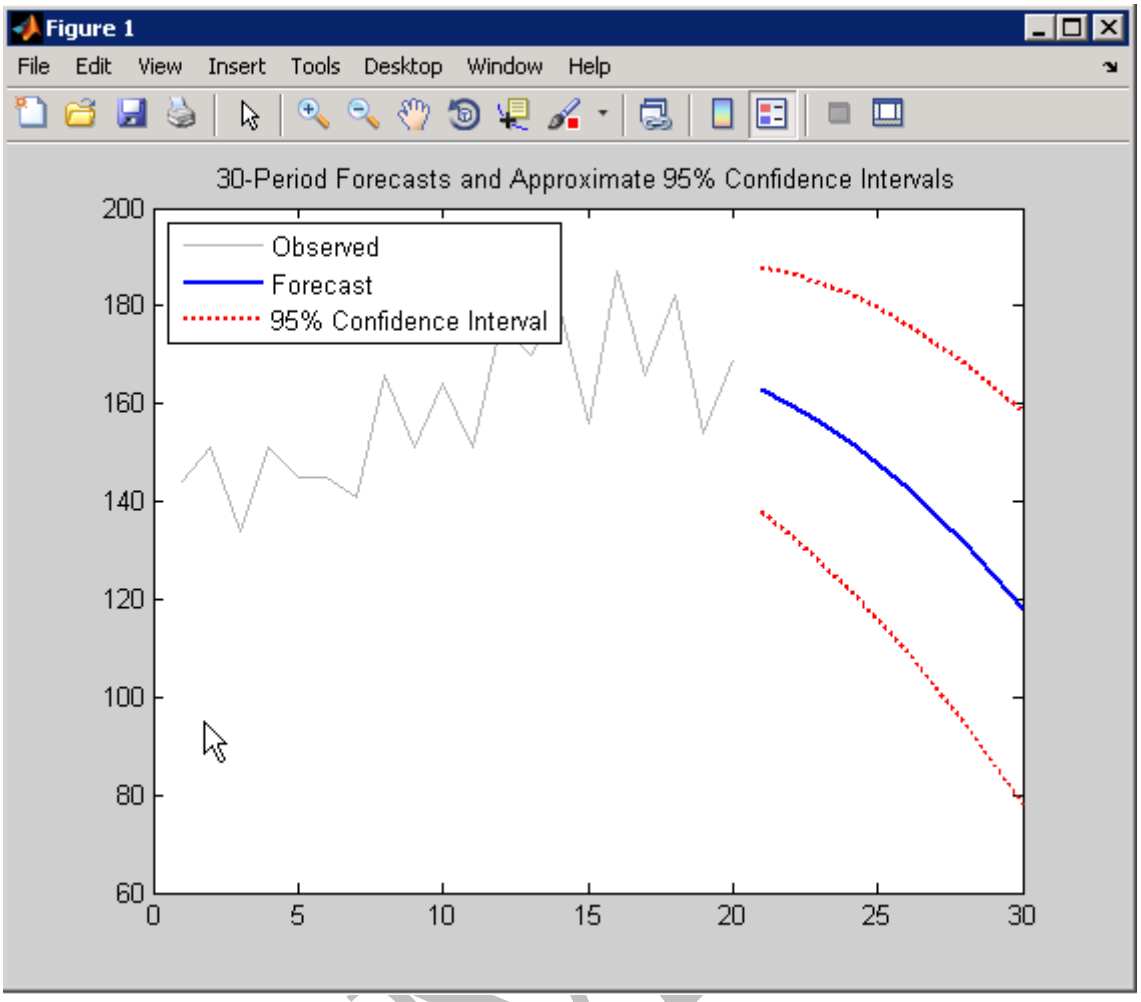
EDITOR PUBLISH VIEW

```
1  clear all;close all;clc
2  arima(2,0,0)
3  Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4  Mdl=arima(0,2,2);
5  EstMdl=estimate(Mdl,Y')
6  ForecastPeriods=10;
7  [YF YMSE]=forecast(EstMdl,ForecastPeriods,'Y0',Y');
8
9  plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals
```

MPLE


Command Window
ARIMA(0,2,2) Model:
Conditional Probability Distribution: Gaussian
Parameter Value Standard Error t Statistic
Constant -0.473727 0.450129 -1.05242
MA(1) -1.62926 0.198806 -8.19521
MA(2) 0.640409 0.252031 2.54099
Variance 164.673 82.2582 2.00191
EstMdl =
ARIMA(0,2,2) Model:
Distribution: Name = 'Gaussian'
P: 2
D: 2
Q: 2
Constant: -0.473727
AR: {}
SAR: {}
MA: (-1.62926 0.640409) at Lags [1 2]
SMA: {}
Variance: 164.673

SAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
200
180
160
140
120
100
80
60
0
5
10
15
20
25
30

SAM

```
Command Window
  BackTesting =

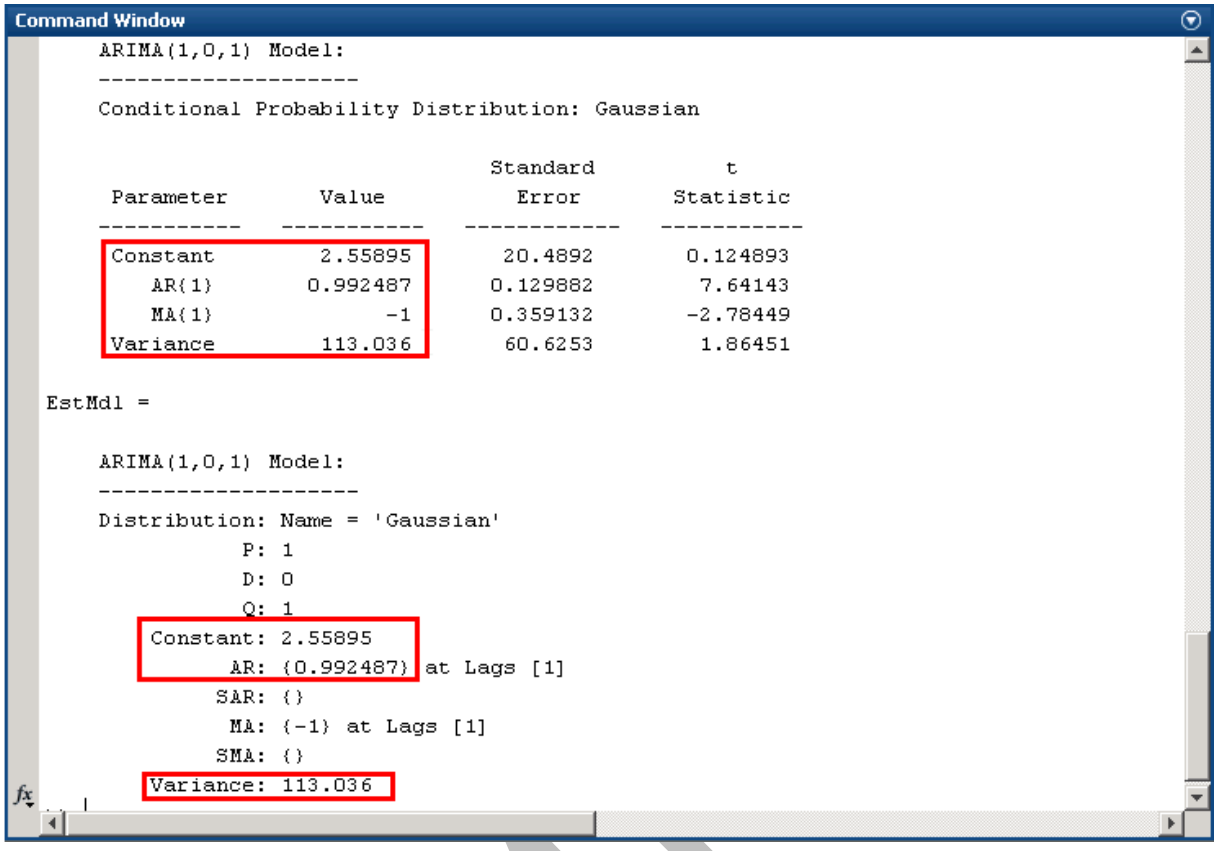
    144.0000
    151.0000
    157.5263
    154.8566
    158.7433
    158.4478
    157.6348
    155.0166
    161.9784
    160.4468
    163.7065
    160.5040
    167.1419
    168.7935
    173.0983
    166.5606
    173.2753
    169.4692
    172.4512
    163.6129
```

Univariate ARMA(1,1)/ARIMA(1,0,1)

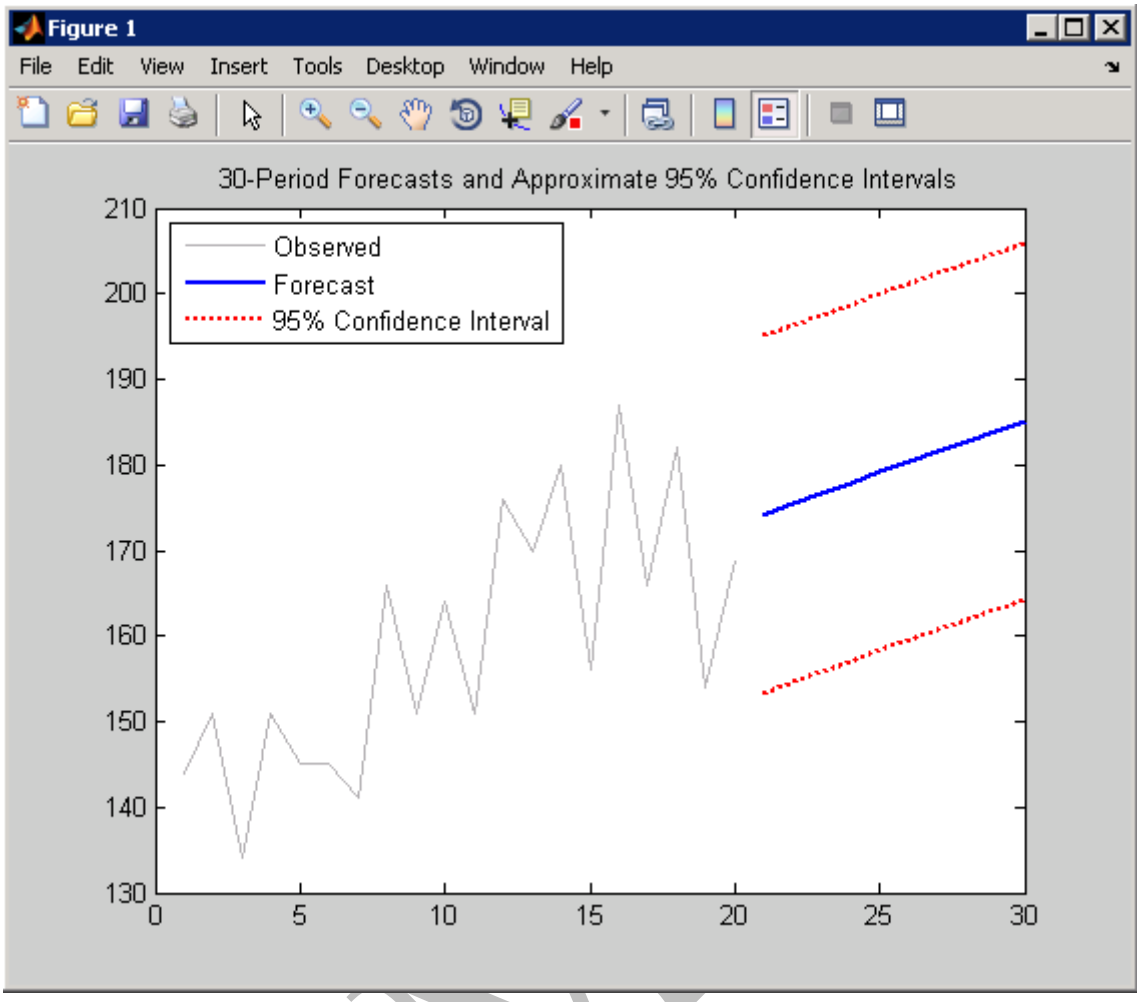
EDITOR PUBLISH VIEW

```
1   clear all;close all;clc
2   arima(2,0,0)
3   Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4   Mdl=arima(1,0,1);
5   EstMdl=estimate(Mdl,Y')
6   ForecastPeriods=10;
7   [YF YMSE]=forecast(EstMdl,ForecastPeriods,'Y0',Y');
8
9   plot(Y,'Color',[.7,.7,.7]);
10  hold on
11  plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12  plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
13  plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
14  legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15  title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17  %this part is to obtain values for backtesting
18  residuals=infer(EstMdl,Y');
19  BackTesting=Y'-residuals
```

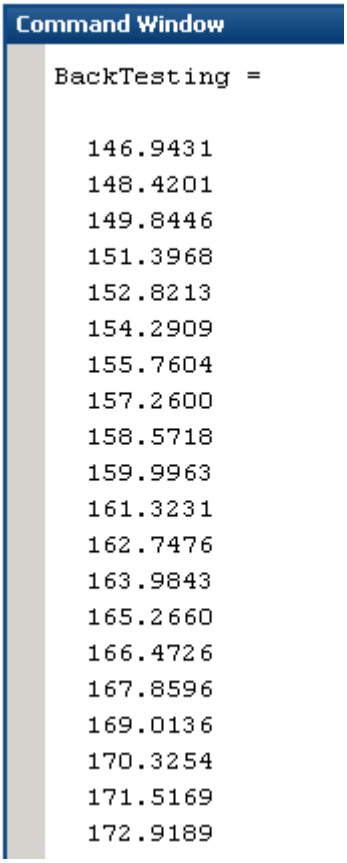
MPLE


Command Window
ARIMA(1,0,1) Model:
Conditional Probability Distribution: Gaussian
Parameter Value Standard Error t Statistic
Constant 2.55895 20.4892 0.124893
AR{1} 0.992487 0.129882 7.64143
MA{1} -1 0.359132 -2.78449
Variance 113.036 60.6253 1.86451
EstMdl =
ARIMA(1,0,1) Model:
Distribution: Name = 'Gaussian'
P: 1
D: 0
Q: 1
Constant: 2.55895
AR: {0.992487} at Lags [1]
SAR: {}
MA: {-1} at Lags [1]
SMA: {}
Variance: 113.036

SAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
30-Period Forecasts and Approximate 95% Confidence Intervals
Observed
Forecast
95% Confidence Interval
210
200
190
180
170
160
150
140
130
0
5
10
15
20
25
30

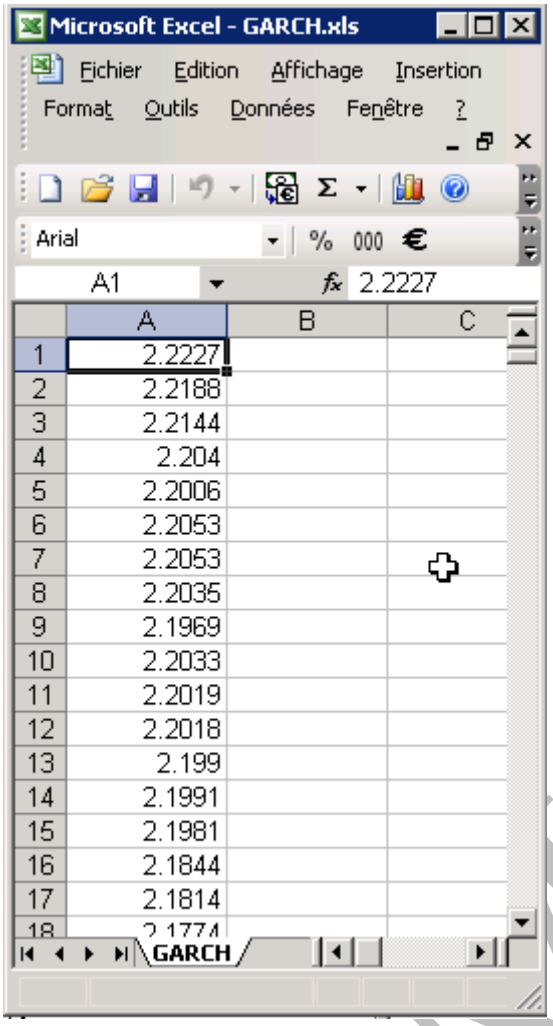
SAM

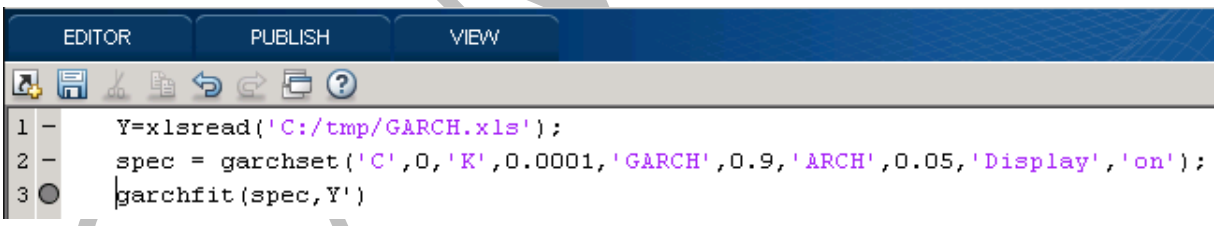


Univariate GARCH(1,1)

1267 rows

SAMPLE


Microsoft Excel - GARCH.xls
Fichier Edition Affichage Insertion
Format Outils Données Fenêtre ?
Arial
A1
2.2227
A B C
1 2.2227
2 2.2188
3 2.2144
4 2.204
5 2.2006
6 2.2053
7 2.2053
8 2.2035
9 2.1969
10 2.2033
11 2.2019
12 2.2018
13 2.199
14 2.1991
15 2.1981
16 2.1844
17 2.1814
GARCH


EDITOR
PUBLISH
VIEW
1 Y=xlsread('C:/tmp/GARCH.xls');
2 spec = garchset('C',0,'K',0.0001,'GARCH',0.9,'ARCH',0.05,'Display','on');
3 garchfit(spec,Y')

SAMPLE

```
Command Window

Boundary constraints active: standard errors may be inaccurate.

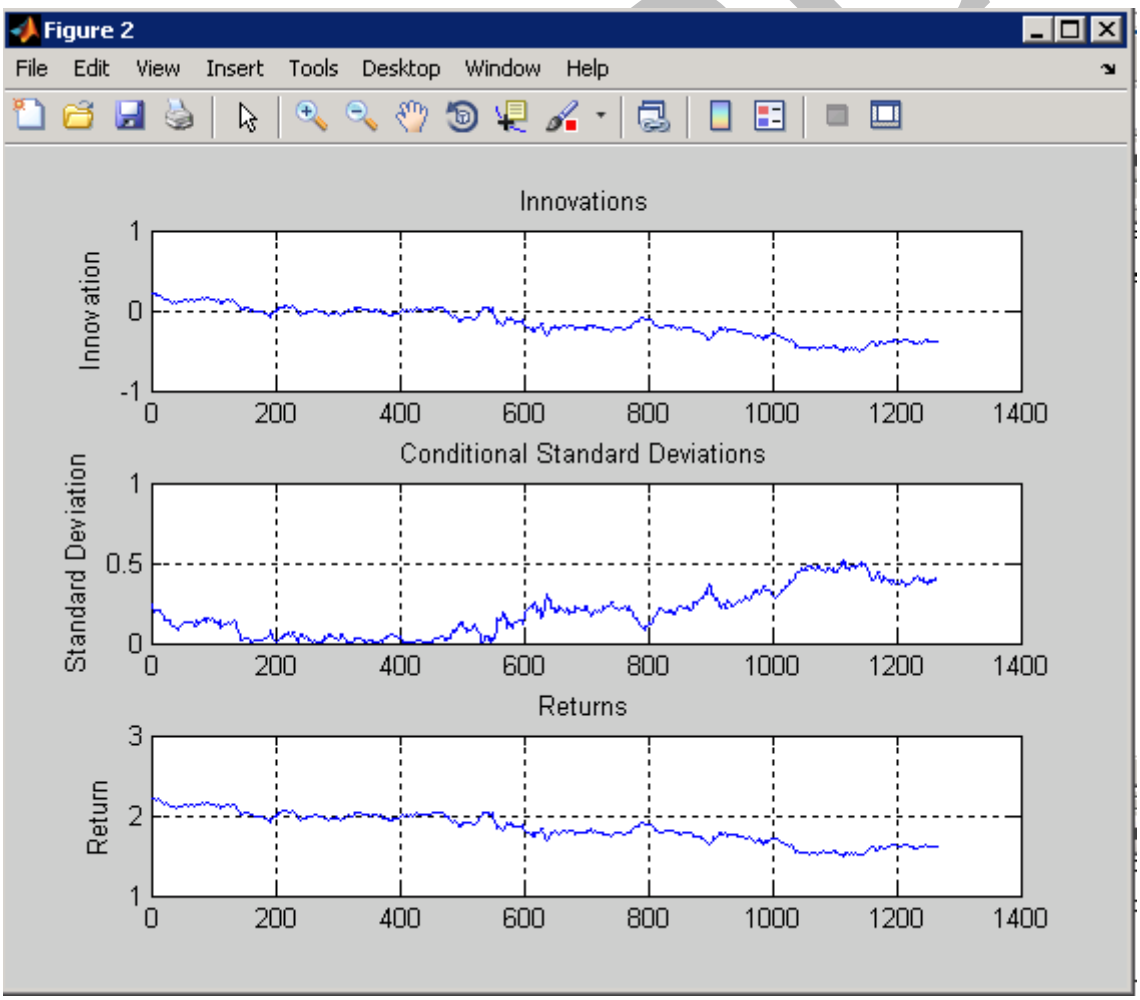
  Mean: ARMAX(0,0,0); Variance: GARCH(1,1)

  Conditional Probability Distribution: Gaussian
  Number of Model Parameters Estimated: 4

                              Standard          T
  Parameter       Value         Error       Statistic
 -----------   -----------   ------------   -----------
           C     2.0018        0.00084902    2357.7961
           K     5.1299e-05    7.1504e-06       7.1743
   GARCH(1)      0.079821      0.05266          1.5158
    ARCH(1)      0.92018       0.17954          5.1251

  Log Likelihood Value: 925.077
```

Looking at the results it seems that MATLAB use for ε_t the difference between two consecutive return. The result is thus very different from R.



EDITOR PUBLISH VIEW

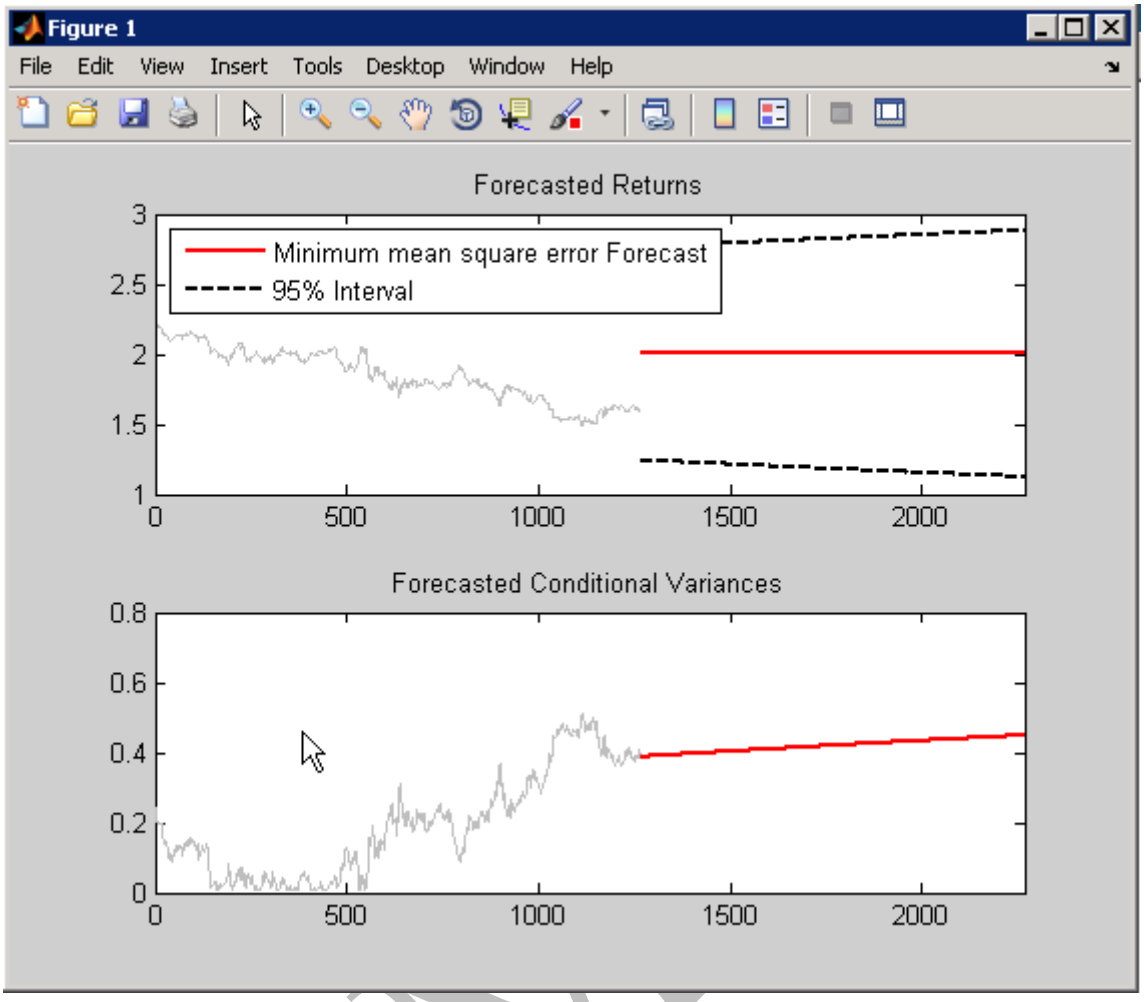
```
Y=xlsread('C:/tmp/GARCH.xls');
N=length(Y);
spec = garchset('C',0,'K',0.0001,'GARCH',0.9,'ARCH',0.05,'Display','on');
[coeff,error]=garchfit(spec,Y');
[e0,s0]=garchinfer(coeff,Y');
[sp,yp,~,ymse] = garchpred(coeff,Y',1000);

upper = yp + 1.96*ymse;
lower = yp - 1.96*ymse;

subplot(2,1,1)
plot(Y,'Color',[.75,.75,.75])
hold on
h1 = plot(N+1:N+1000,yp,'r','LineWidth',2);
h2 = plot(N+1:N+1000,upper,'k--','LineWidth',1.5);
plot(N+1:N+1000,lower,'k--','LineWidth',1.5)
xlim([0,N+1000])
title('Forecasted Returns')
legend([h1,h2],'Minimum mean square error Forecast','95% Interval',...
        'Location','NorthWest')
hold off

subplot(2,1,2)
plot(s0,'Color',[.75,.75,.75])
hold on
plot(N+1:N+1000,sp,'r','LineWidth',2)
xlim([0,N+1000])
title('Forecasted Conditional Variances')
hold off
```

SAM



We don't get the same chart as calculated by hand or as obtained with R...

SAMPLE

Applications (without Financial Toolbox)

The codes available here do not contain any programmatic essentials such as error checking and they are not optimized for memory efficiency or speed.

Bond portfolio with duration/convexity targets

$$\sum_{i=1}^{n} D_i^* w_i = D_p^*$$

$$\sum_{i=1}^{n} CVX_i w_i = CVX_p$$

$$\sum_{i=1}^{n} w_i = 1$$

EDITOR PUBLISH VIEW

```
1   clear all;clc
2   settle='2007-08-28';
3   maturities=['2012-06-15' ; '2017-10-31' ; '2027-03-01'];
4   couponRates=[0.07 ; 0.06 ; 0.08];
5   yields=[0.06 ; 0.07 ; 0.075];
6
7   %we compute durations and convexities
8   durations=bnddury(yields, couponRates, settle, maturities);
9   convexities=bndconvy(yields, couponRates, settle, maturities);
10
11  %we compute weights by solvie the simple linear system
12  A=[durations'
13      convexities'
14      1 1 1];
15  %we want a duration of 10 years with a convexity of 160
16  %and sum of weight = 1
17  b=[10 160 1];
18  %simple linear algebra
19  weights=A\b'
```

Bond portfolio with duration/convexity targets and maximization of average yield

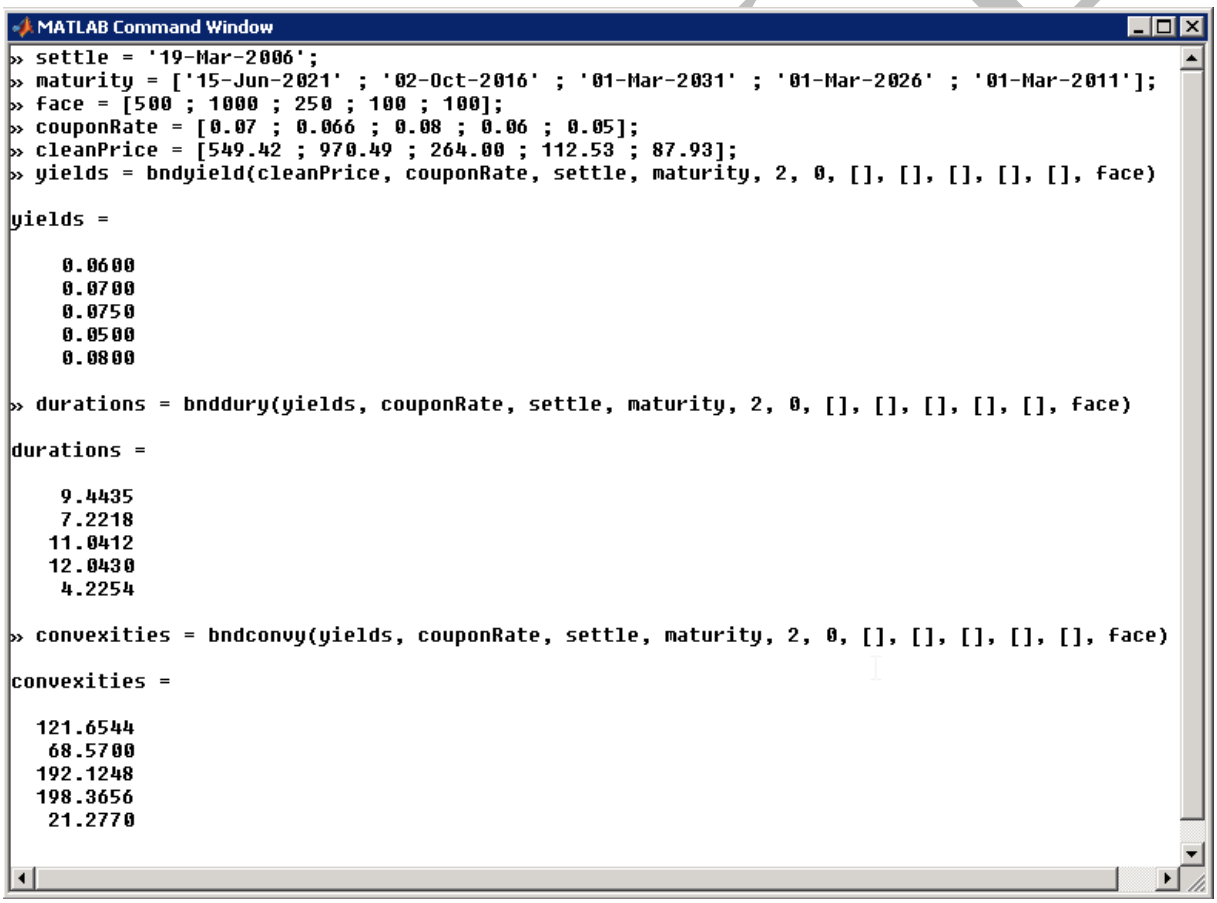
$$\max\ z = \sum_{i=1}^{n} Y_i\ w_i$$

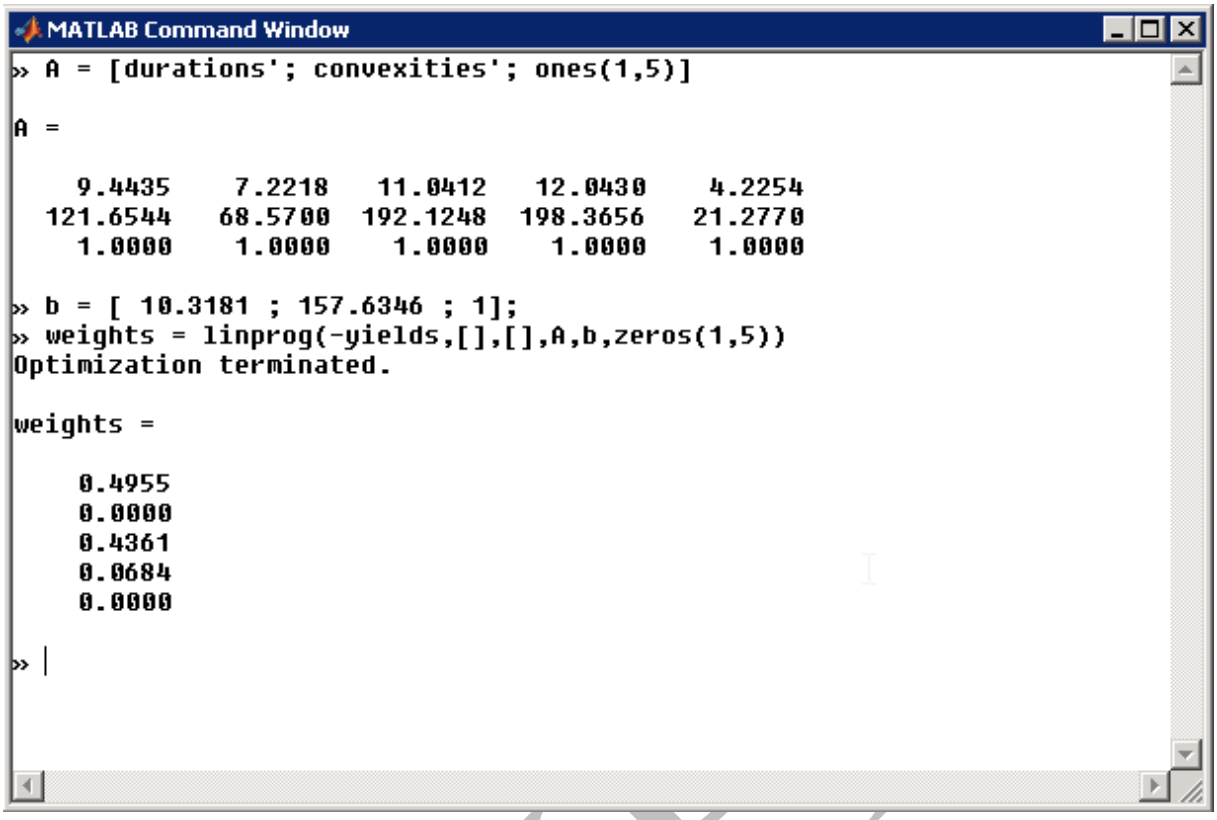
$$\sum_{i=1}^{n} D_i^* w_i = D_p^*$$

$$\sum_{i=1}^{n} CVX_i w_i = CVX_p$$

$$\sum_{i=1}^{n} w_i = 1$$

$$w_i \geq 0$$



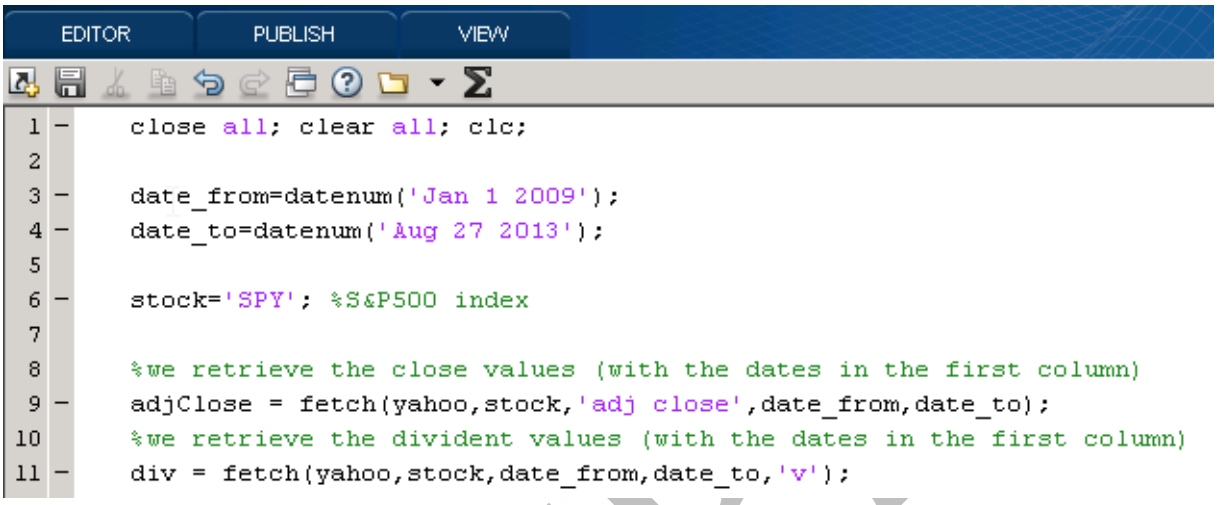

MATLAB Command Window
» A = [durations'; convexities'; ones(1,5)]
A =
9.4435 7.2218 11.0412 12.0430 4.2254
121.6544 68.5700 192.1248 198.3656 21.2770
1.0000 1.0000 1.0000 1.0000 1.0000
» b = [10.3181 ; 157.6346 ; 1];
» weights = linprog(-yields,[],[],A,b,zeros(1,5))
Optimization terminated.
weights =
0.4955
0.0000
0.4361
0.0684
0.0000
»

SAMPLE

Backtesting long position buying-holding and selling SPY

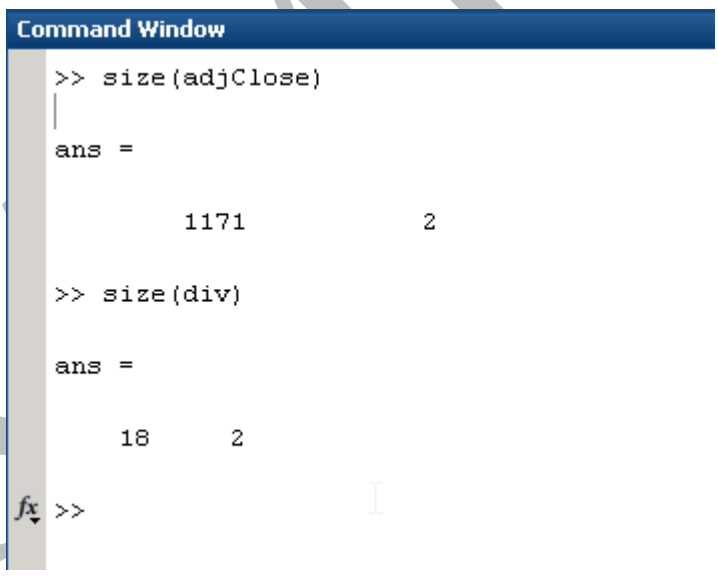
It is a well known fact that one may expect the drop of stock price within a short timeframe (e.g. a few days) after the day when the stock's dividends have been announced.

Thus, the purpose is to test and find the best time period of buying-holding-and-selling SPY (or any other stock paying dividends) to make a profit every time.

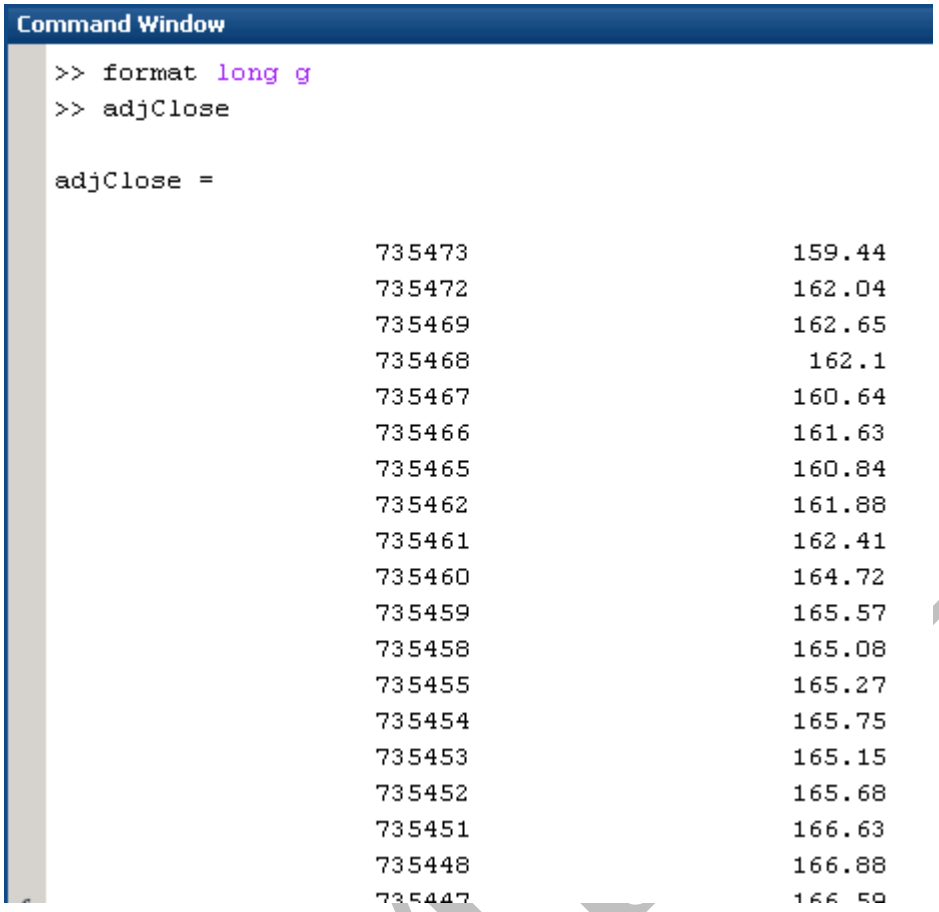



```
1 -    close all; clear all; clc;
2
3 -    date_from=datenum('Jan 1 2009');
4 -    date_to=datenum('Aug 27 2013');
5
6 -    stock='SPY'; %S&P500 index
7
8      %we retrieve the close values (with the dates in the first column)
9 -    adjClose = fetch(yahoo,stock,'adj close',date_from,date_to);
10     %we retrieve the divident values (with the dates in the first column)
11 -   div = fetch(yahoo,stock,date_from,date_to,'v');
```

At this point, we have:



Explicitly:


Command Window
>> format long g
>> adjClose
adjClose =
735473 159.44
735472 162.04
735469 162.65
735468 162.1
735467 160.64
735466 161.63
735465 160.84
735462 161.88
735461 162.41
735460 164.72
735459 165.57
735458 165.08
735455 165.27
735454 165.75
735453 165.15
735452 165.68
735451 166.63
735448 166.88

And:

SAMPLE

```
Command Window
>> div

div =

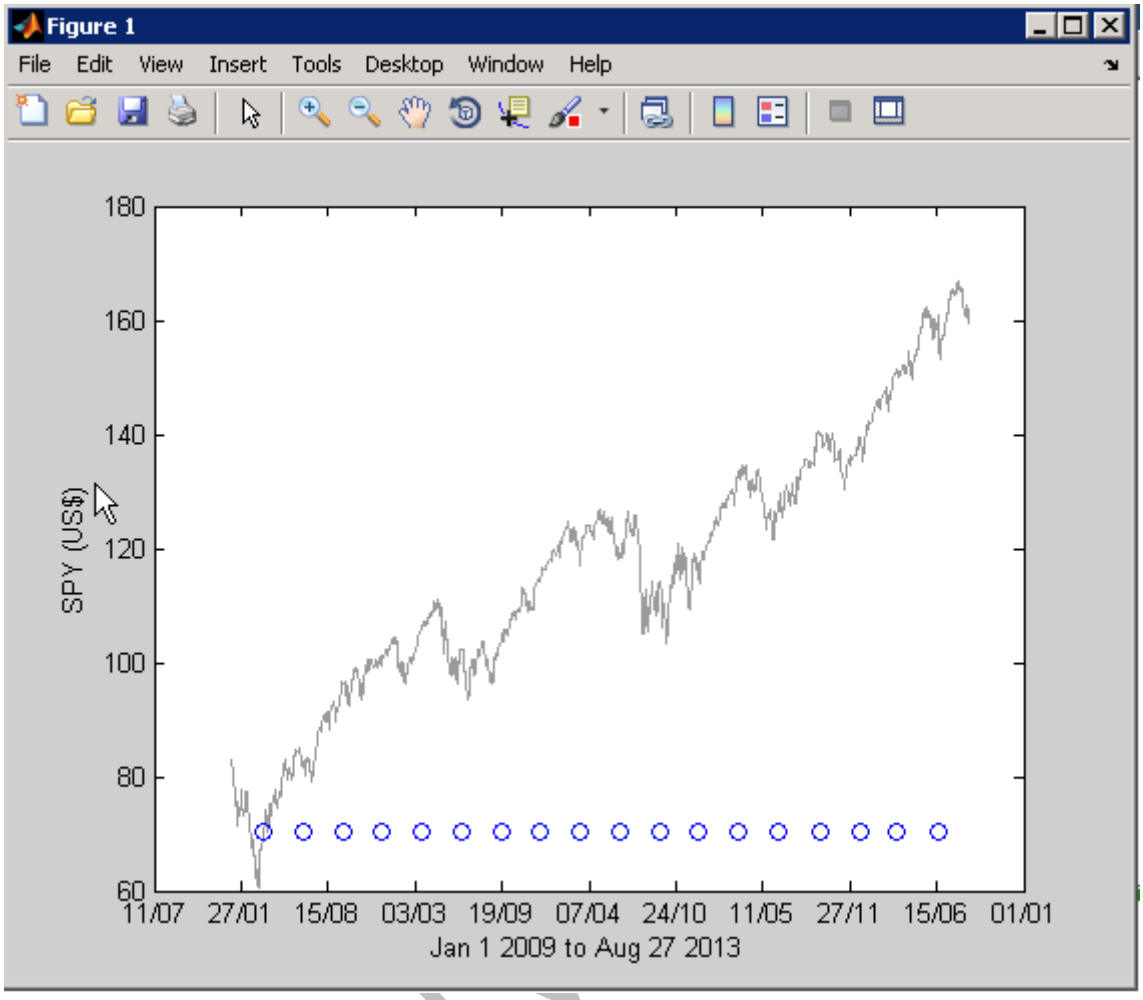
                735406                   0.839
                735308                   0.694
                735224                   1.022
                735133                   0.779
                735035                   0.688
                734944                   0.614
                734853                    0.77
                734762                   0.625
                734671                   0.628
                734580                   0.553
                734489                   0.653
                734398                   0.602
                734307                   0.531
                734216                    0.48
                734125                    0.59
                734034                   0.508
                733943                   0.518
                733852                   0.561
```

And now we plot in these informations:

```
EDITOR  PUBLISH  VIEW

 1 -    close all; clear all; clc;
 2
 3 -    date_from=datenum('Jan 1 2009');
 4 -    date_to=datenum('Aug 27 2013');
 5
 6 -    stock='SPY'; %S&P500 index
 7
 8      %we retrieve the close values (with the dates in the first column)
 9 -    adjClose = fetch(yahoo,stock,'adj close',date_from,date_to);
10      %we retrieve the divident values (with the dates in the first column)
11 -    div = fetch(yahoo,stock,date_from,date_to,'v');
12
13      % We plot adjusted close price of and mark days when dividends
14      % have been announced
15 -    plot(adjClose(:,1),adjClose(:,2),'color',[0.6 0.6 0.6])
16 -    datetick('x','dd/mm','keepticks');
17 -    hold on;
18      %the offset to draw the dividends is empirical and just for esthetical purposes
19 -    plot(div(:,1),min(adjClose(:,2))+10,'ob');
20 -    ylabel('SPY (US$)');
21 -    xlabel('Jan 1 2009 to Aug 27 2013');
```

So we get:



Having the data ready for backtesting, let's look for the most profitable period of time of buying-holding-and-selling SPY assuming that we buy SPY one day after the dividends have been announced (at the market price), and we hold for *dt* days (here, tested to be between 1 and 40 trading days):

SAMPLE

```
21 -    xlabel('Jan 1 2009 to Aug 27 2013');
22
23      %find the most profitable period of holding SPY (long position)
24 -    neg=[];
25 -    for dt=1:40
26 -        buy=[]; sell=[];
27 -        for i=1:size(div,1)
28              %find the dates (rows) of adjClose when the dividends have been announced
29 -            r=find(adjClose(:,1)==div(i,1));
30              %mark the corresponding SPY price with blue circle marker
31 -            hold on; plot(adjClose(r,1),adjClose(r,2),'ob');
32              %assume you buy long SPY next day at the market price (close price)
33 -            buy=[buy; adjClose(r-1,1) adjClose(r-1,2)];
34              % assume you sell SPY in 'dt' days after you bought SPY at the market
35              % price (close price)
36 -            sell=[sell; adjClose(r-1-dt,1) adjClose(r-1-dt,2)];
37 -        end
38          %calculate profit-and-loss (PnL) of each trade (excluding transaction costs)
39 -        PnL=sell(:,2)./buy(:,2)-1;
40          %summarize the results
41 -        neg=[neg; dt sum(PnL<0) sum(PnL<0)/length(PnL)];
42 -    end
```

So we get:

Command Window

```
>> sortrows(neg,3)

ans =

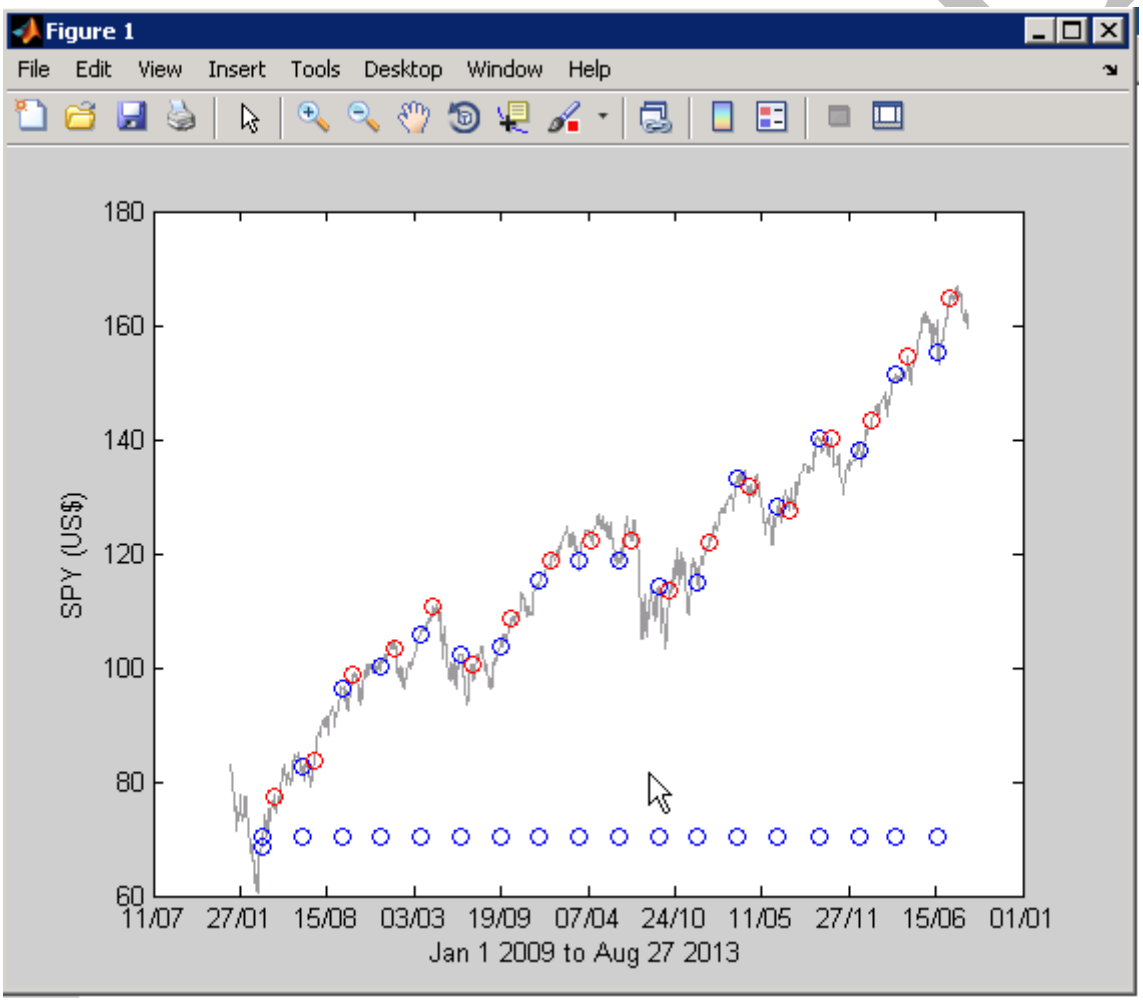
    18     2     0.111111111111111
    17     3     0.166666666666667
    19     3     0.166666666666667
    24     3     0.166666666666667
     9     4     0.222222222222222
    14     4     0.222222222222222
    20     4     0.222222222222222
    21     4     0.222222222222222
    23     4     0.222222222222222
    25     4     0.222222222222222
    28     4     0.222222222222222
    29     4     0.222222222222222
    13     5     0.277777777777778
    15     5     0.277777777777778
    16     5     0.277777777777778
    22     5     0.277777777777778
    27     5     0.277777777777778
    30     5     0.277777777777778
    31     5     0.277777777777778
    33     5     0.277777777777778
```

We see that the best strategy seems to take a long position on 18 days.

Now we make a plot running the whole script from the beginning:

```
42 -    end
43
44 -    hold on;
45 -    for i=1:size(div,1)
46 -        r=find(adjClose(:,1)==div(i,1));
47          % mark the corresponding SPY price with blue circle marker
48 -        hold on; plot(adjClose(r-18,1),adjClose(r-18,2),'or');
49 -    end
```

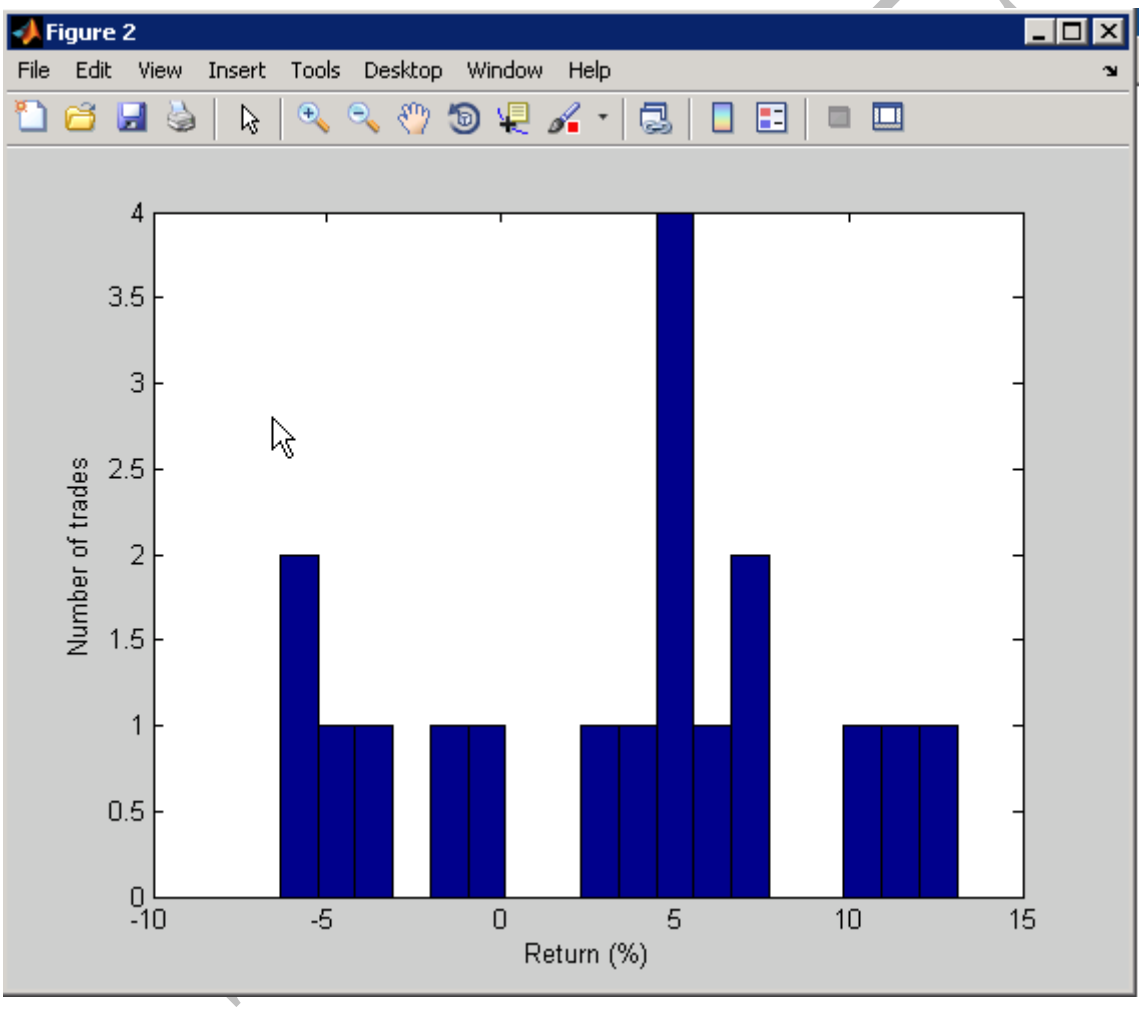
So we get have in blue the days paying the dividend and in reg the 18 days after where it could be good to sell:



And finally:

```
48 -        hold on; plot(adjClose(r-18,1),adjClose(r-18,2),'or');
49 -    end
50
51      %The complete distribution of profit and losses from all trades
52      %can be obtained in the following way:
53 -    figure(2);
54 -    hist(PnL*100,length(PnL))
55 -    ylabel('Number of trades')
56 -    xlabel('Return (%)')
57
```

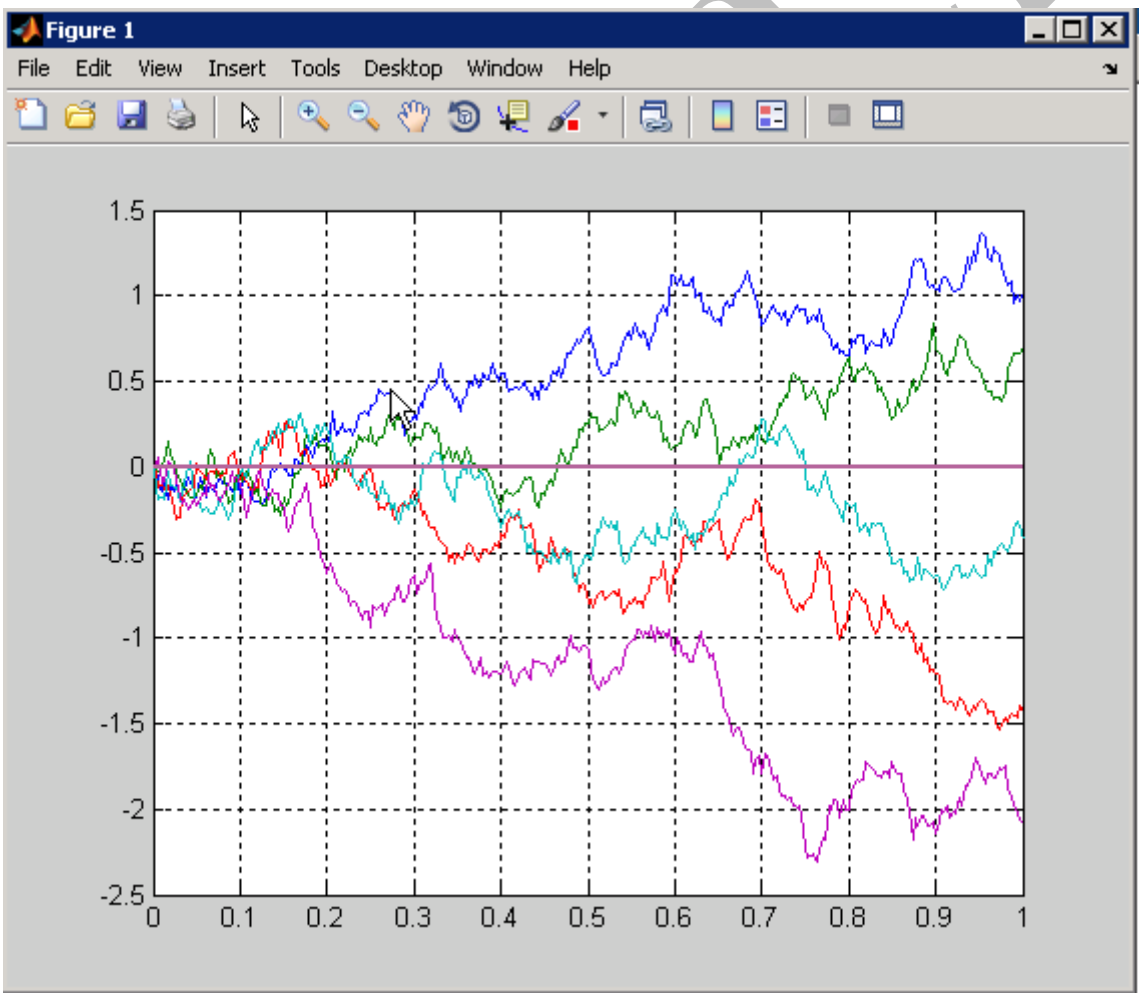
That give us:



Brownian motion and Wiener Process

First we plot a few brownian motions:

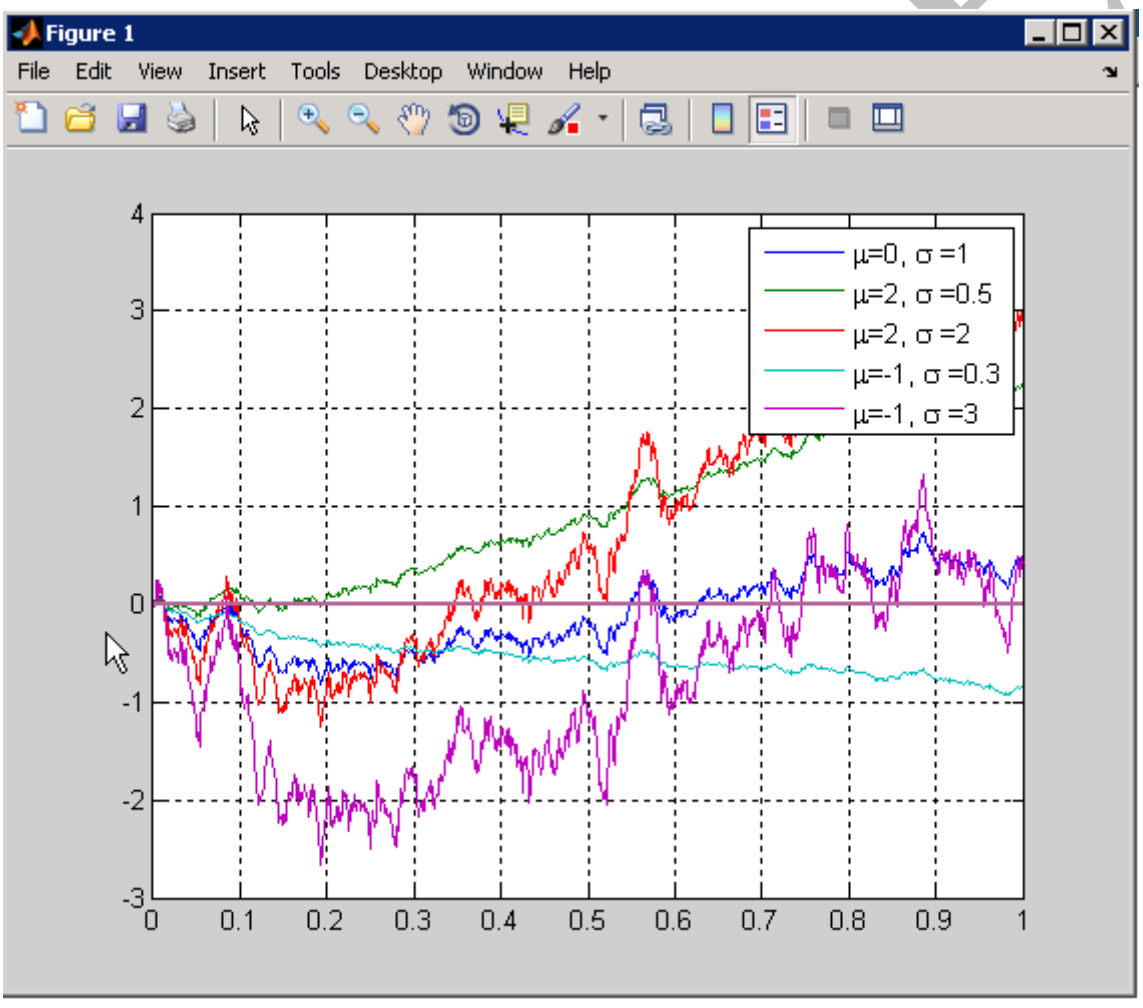
```
m=5;
n=300;
t=linspace(0,1,n+1);
h=diff(t(1:2));
dw=sqrt(h)*randn(n,m);
w=cumsum([zeros(1,m);dw]);
plot(t,w);
grid on;
hy=graph2d.constantline(0,'color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hy,'y');
```



And now a few Wiener processes with different drifts and volatilities:

EDITOR PUBLISH VIEW

```
n=1000;
t=linspace(0,1,n+1);
h=diff(t(1:2));
dw=sqrt(h)*randn(n,1);
w=cumsum([0;dw]);
plot(t,t'*[0 2 2 -1 -1]+w*[1 0.5 2 0.3 3]);
legend('\mu=0, \sigma =1','\mu=2, \sigma =0.5','\mu=2, \sigma =2',...
    '\mu=-1, \sigma =0.3','\mu=-1, \sigma =3');
grid on;
hy=graph2d.constantline(0,'color',[0.7 0.4 0.6],'LineWidth',2);
changedependvar(hy,'y');
```



Risk-return asset pricing models

Capital Asset Pricing Model (CAPM)

EDITOR PUBLISH VIEW

```
clear all;close all;clc
ra=[-0.15,0.05,-0.43,0.79,0.32];
rb=[0.29,0.18,0.24,0.25,0.17];
rc=[0.38,0.63,0.46,0.36,-0.57];
PfM=[0.45,0.61,0.54,0.29,0.12];
Rf=repmat(0,1,length(PfM));
Returns=[ra',rb',rc',PfM'];
[m,n]=size(Returns);
%compute the excess return
exreturn = zeros(m,n);
for i = 1:m
    exreturn(i,:)=Returns(i,:)- Rf(i);
end
%regression with each risky asset on the portfolio, note that the
%last columns of the exreturn matrix is  market premium data
for i= 1:n
    CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Covariance matrix
    % beta = cov(x,y)/var(x);
    Beta(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
    % Alpha = return - (rf + beta(E(rm)-rf))
    Alpha(i)= mean(exreturn(:,i))-(mean(Rf)+ Beta(i)*mean(exreturn(:,n)));
    % rt = rf + beta(E(rm)-rf)
    Rt(i) = mean(Rf) +  Beta(i)*mean(exreturn(:,n));
end
capm_Alpha = Alpha
capm_Beta = Beta
capm_return = Rt
```

```
Command Window

capm_Alpha =

    0.7186    0.2041   -0.6029         0


capm_Beta =

   -1.4990    0.0545    2.1265    1.0000


capm_return =

   -0.6026    0.0219    0.8549    0.4020

fx >>
```

Three-factor Fama-French model (FFM)

EDITOR PUBLISH VIEW

```
4     %ER: Excess Return (Assets Return - Risk Free Return)
5     %also known as "Risk Adjusted Return".
6 -   ER=[4.7 -10.26 26.09 0.06 13.68 16.74 14.65 20.63 15 -15.81 -23.88];
7     %SMB: "Small [market capitalization] Minus Big"
8     %1/3(Small Value+Small Neutral+Small Growth)-1/3(Big Value+Big Neutral+Big Growth)
9 -   SMB=[-9.61 -12.66 4.58 7.81 -9.67 -2.14 -13.36 -15.78 14.86 -6.24 0.53];
10    %HML: "High [book-to-market ratio] Minus Low"
11    %1/2(Small Value+Big Value)-1/2(Small Growth+Big Growth)
12 -  HML=[-6.4 -2.51 23.22 5.4 -5.82 1.96 13.1 -0.52 -22.2 28.17 25.27];
13    %Risk free return
14 -  Rf=[5.6 3.5 2.9 3.91 5.6 5.2 5.25 4.85 4.69 5.88 3.85];
15
16 -  format bank
17 -  FF3=[year', ER', SMB', HML',Rf'];
18 -  printmat(FF3,'Raw Data:',num2str(1:size(FF3)),'Date (Rmt-rf) SMB HML rf')
19
20 -  AssetReturn1=[0.010 0.022 0.074 0.02 0.009 0.042 0.045 0.069 0.079 0.077 0.087];
21 -  AssetReturn2=[0.057 0.026 0.040 0.021 0.017 0.058 0.063 0.053 0.084 0.053 0.06];
22 -  Portfolio=[AssetReturn1' AssetReturn2'];
23 -  printmat(Portfolio,'Portfolio',num2str(1:size(Portfolio)),'Asset(1) Asset(2)')
```

Command Window

```
Raw Data: =
                    Date      (Rmt-rf)          SMB          SMH           rf
     1    1991.00000      4.70000     -9.61000     -6.40000      5.60000
     2    1992.00000    -10.26000    -12.66000     -2.51000      3.50000
     3    1993.00000     26.09000      4.58000     23.22000      2.90000
     4    1994.00000      0.06000      7.81000      5.40000      3.91000
     5    1995.00000     13.68000     -9.67000     -5.82000      5.60000
     6    1996.00000     16.74000     -2.14000      1.96000      5.20000
     7    1997.00000     14.65000    -13.36000     13.10000      5.25000
     8    1998.00000     20.63000    -15.78000     -0.52000      4.85000
     9    1999.00000     15.00000     14.86000    -22.20000      4.69000
    10    2000.00000    -15.81000     -6.24000     28.17000      5.88000
    11    2001.00000    -23.88000      0.53000     25.27000      3.85000

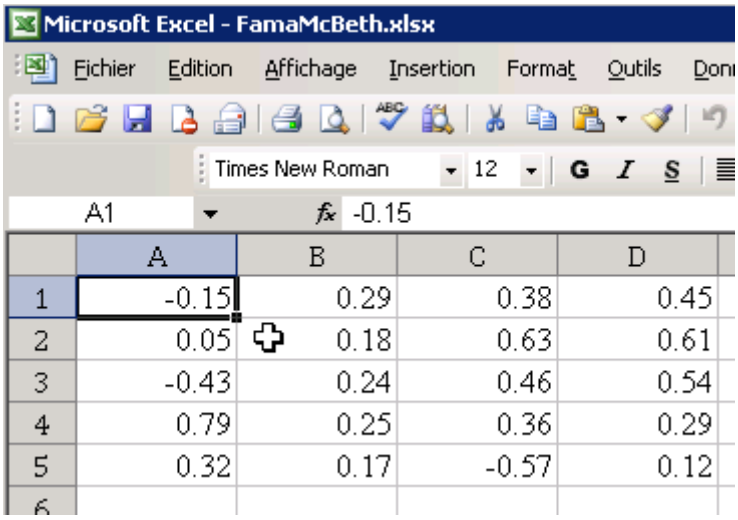
fx >>
```

Command Window

```
Portfolio =
              Asset(1)      Asset(2)
     1        0.01000       0.05700
     2        0.02200       0.02600
     3        0.07400       0.04000
     4        0.02000       0.02100
     5        0.00900       0.01700
     6        0.04200       0.05800
     7        0.04500       0.06300
     8        0.06900       0.05300
     9        0.07900       0.08400
    10        0.07700       0.05300
    11        0.08700       0.06000
```

```
24
25 -    FF3(:,1)=[]; %remove date data
26 -    FF3=FF3/100; %format data in %
27 -    [~,last_col]=size(FF3);
28 -    FF3(:,last_col)=[]; %remove rf
29 -    [N ,M] = size(Portfolio);
30 -    for i = 1:M
31          %compute excess return for all asset in portfolio
32 -        exreturn(:,i) = Portfolio(:,i) - Rf';
33          %get beta coefficient using multivariate regression function
34          %such as: (rt-rf) = b(Rm-rf) + bs(SMB) + bv(HML) + alpha
35 -        coef(:,i) = regress(exreturn(:,i),FF3);
36 -        expReturn(:,i) = mean(Rf') + coef(1,i)*mean(FF3(:,1))...
37              +coef(2,i)*mean(FF3(:,2))+coef(2,i)*mean(FF3(:,3));
38 -    end
39
40 -    for i = 1:M
41 -    fprintf('Asset(%d)>> beta(i): %0.2f  b(SMB): %0.2f  b(HML): %0.2f 3ffExp_Return: %0.2f Alpha: %0.2f\n',...
42          i,coef(1,i),coef(2,i),coef(3,i),expReturn(1,i),mean(Portfolio(:,i))-expReturn(1,i))
43 -    end
```

Fama-MacBeth



	A	B	C	D
1	-0.15	0.29	0.38	0.45
2	0.05	0.18	0.63	0.61
3	-0.43	0.24	0.46	0.54
4	0.79	0.25	0.36	0.29
5	0.32	0.17	-0.57	0.12
6				

EDITOR PUBLISH VIEW

```
%load a data contains all risky asset
%the last column is for market index (market return)
clear all;close all;clc
Returns = xlsread('E:\FamaMcBeth.xlsx');
List = {'T1','T2','T3','MI'};
disp(List)
disp(Returns)
%the T-bill rate to daily rate (for this example it is suppose to be fix )
Rf = repmat(0,1,length(Returns))*0.02;
[m,n]=size(Returns);
%compute the excess return
exreturn = zeros(m,n);
for i = 1:m
    exreturn(i,:)=Returns(i,:)- Rf(i);
end
%simple CAPM
%regression with each risky asset on the portfolio, note that the
%last columns of the exreturn matrix is  market premium data
for i= 1:n
    CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Convriance matrix
    % beta = cov(x,y)/var(x);
    betacapm(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
    % rt = rf + beta(E(rm)-rf)
    Rt(i) = mean(Rf) +  betacapm(i)* mean(exreturn(:,n));
end
Beta=zeros(m-3,n);%lag 3 years
%Fama McBeth Step 1 :For each of the N securities included in the sample,
%estimate the beta
%(The first step involves a set of 4 years regressions equal in number to
%the number of assets)
```

```
for j = 1 :n
    for i= 1:(m-3)
        % beta = cov(x,y)/var(x);
        Beta(i,j)= regress(exreturn(i:(i+3),j),exreturn(i:(i+3),n));
    end
end
Beta
%Fama McBeth Step 2 : run a cross sectional regression for every t.That is,
%for period t, estimate risk premium
%(The second stage is a set of regressions equal in number to the number of time periods)
%since Fama-Mcbeth empirical CAPM is : exreturn = premium*beta + intercept
premium = zeros(m-3,n);
intercept = zeros(m-3,n);
for j =1:n
for i =1:(m-3)
 premium(i,j) = regress(exreturn(i:(i+3),j),(ones(1,4)*Beta(i,j))');
 intercept(i,j) = exreturn(i+3,j)- premium(i,j)*Beta(i,j);
end
end
premium
%empirical estimated premium
empirical_premium = mean(premium)
%theorical premium
theorical_premium = mean(exreturn)
%empirical average CAPM
rt = mean(Rf) + mean(premium)* mean(Beta)' + mean(intercept);
emprirical_capm = rt
%theorical CAPM
theorical_capm = Rt'
```

Command Window

```
    'T1'     'T2'     'T3'     'MI'

   -0.1500    0.2900    0.3800    0.4500
    0.0500    0.1800    0.6300    0.6100
   -0.4300    0.2400    0.4600    0.5400
    0.7900    0.2500    0.3600    0.2900
    0.3200    0.1700   -0.5700    0.1200


Beta =

   -0.0422    0.4655    0.9556    1.0000
    0.0863    0.4360    0.8773    1.0000


premium =

   -1.5404    0.5155    0.4788    0.4725
    2.1140    0.4817    0.2508    0.3900


empirical_premium =

    0.2868    0.4986    0.3648    0.4312
```

```
theorical_premium =

    0.1160    0.2260    0.2520    0.4020


emprirical_capm =

    1.4279    0.9816    0.5529    0.7704


theorical_capm =

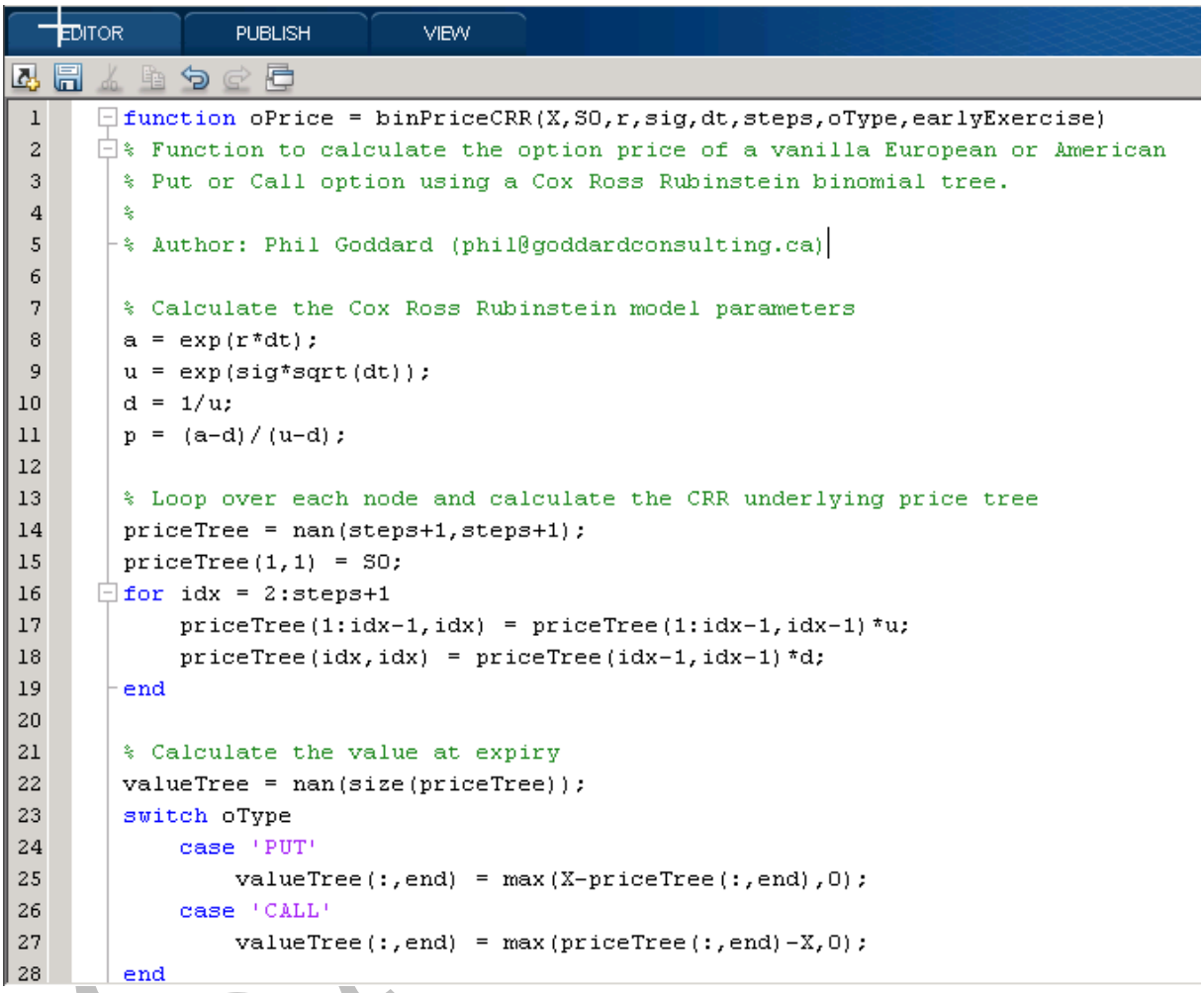
   -0.6026
    0.0219
    0.8549
    0.4020
```

SAMPLE

Options Pricing

CRR binomial option pricing model without Financial Toolbox

Inputs: *X* – strike, S0 - stock price, *r* - risk free interest rate, *sig* – volatility, *dt* - size of time steps, *steps* - number of time steps to calculate, *oType* - must be 'PUT' or 'CALL', *earlyExercise* - true for American, false for European.




```
function oPrice = binPriceCRR(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the option price of a vanilla European or American
% Put or Call option using a Cox Ross Rubinstein binomial tree.
%
% Author: Phil Goddard (phil@goddardconsulting.ca)

% Calculate the Cox Ross Rubinstein model parameters
a = exp(r*dt);
u = exp(sig*sqrt(dt));
d = 1/u;
p = (a-d)/(u-d);

% Loop over each node and calculate the CRR underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end

% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
        valueTree(:,end) = max(priceTree(:,end)-X,0);
end
```

```
29
30      % Loop backwards to get values at the earlier times
31      steps = size(priceTree,2)-1;
32      for idx = steps:-1:1
33          valueTree(1:idx,idx) = ...
34              exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
35              + (1-p)*valueTree(2:idx+1,idx+1));
36          if earlyExercise
37              switch oType
38                  case 'PUT'
39                      valueTree(1:idx,idx) = ...
40                          max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
41                  case 'CALL'
42                      valueTree(1:idx,idx) = ...
43                          max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
44              end
45          end
46      end
47
48      % Output the option price
49      oPrice = valueTree(1);
```

```
Command Window
>> oPrice = binPriceCRR(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.6279

fx >>
```

Drifted CRR binomial option pricing model without Financial Toolbox

Inputs: *X* – strike, S0 - stock price, *r* - risk free interest rate, *sig* – volatility, *dt* - size of time steps, *steps* - number of time steps to calculate, *oType* - must be 'PUT' or 'CALL', *earlyExercise* - true for American, false for European, *drift* – will be the drift...

EDITOR PUBLISH VIEW

```
function oPrice=binPriceCRRDrift(X,S0,r,sig,dt,steps,oType,earlyExercise,drift)
% Author: Phil Goddard (phil@goddardconsulting.ca)

% Calculate the CRR With Drift model parameters
a = exp(r*dt);
u = exp(drift*dt + sig*sqrt(dt));
d = exp(drift*dt - sig*sqrt(dt));
p = (a-d)/(u-d);

% Loop over each node and calculate the underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end

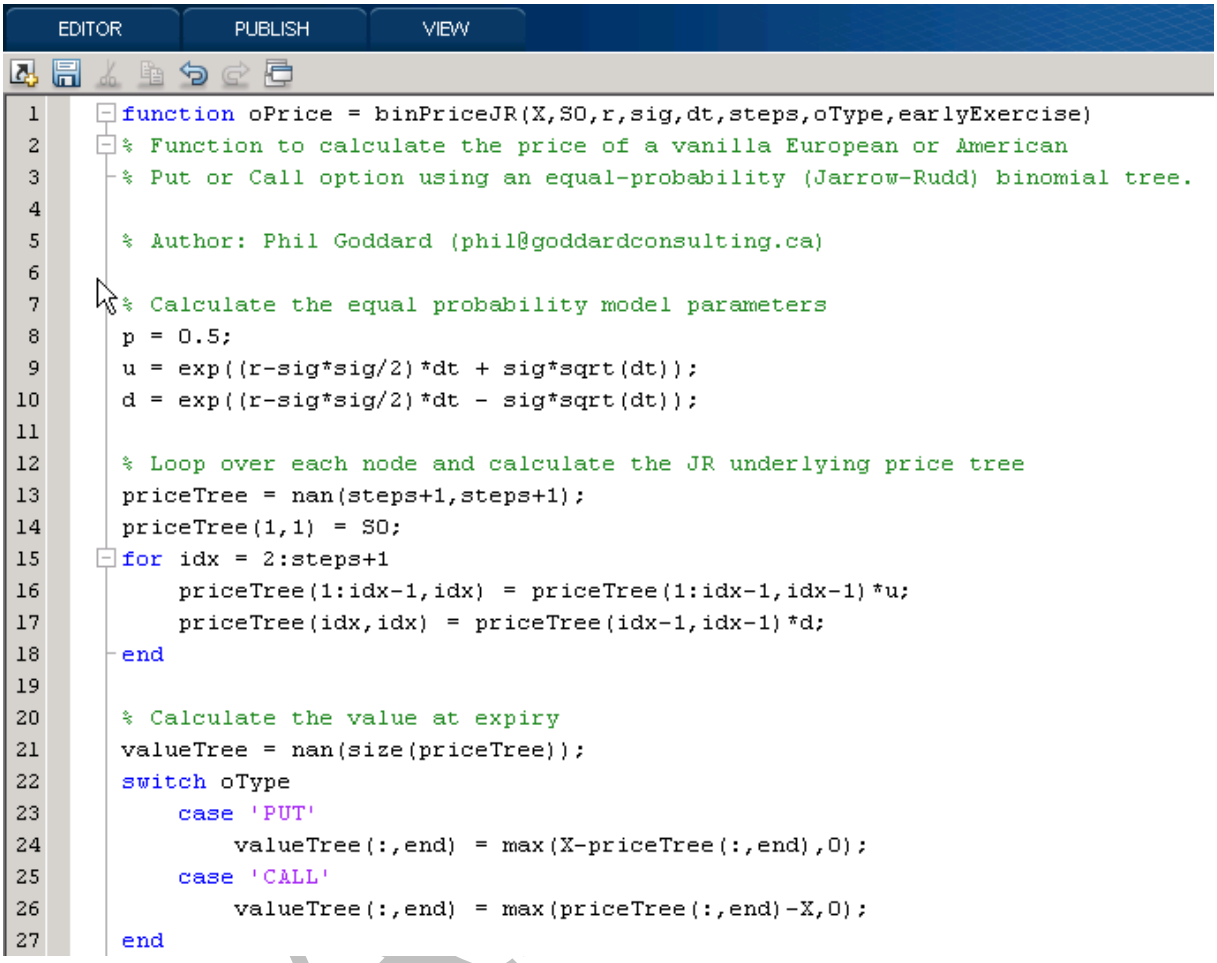
% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
        valueTree(:,end) = max(priceTree(:,end)-X,0);
end


% Loop backwards to get values at the earlier times
steps = size(priceTree,2)-1;
for idx = steps:-1:1
    valueTree(1:idx,idx) = ...
        exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
        + (1-p)*valueTree(2:idx+1,idx+1));
    if earlyExercise
        switch oType
            case 'PUT'
                valueTree(1:idx,idx) = ...
                    max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
            case 'CALL'
                valueTree(1:idx,idx) = ...
                    max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
        end
    end
end

% Output the option price
oPrice = valueTree(1);
```

Jarrow-Rudd binomial option pricing model without Financial Toolbox

Inputs: *X* – strike, S0 - stock price, *r* - risk free interest rate, *sig* – volatility, *dt* - size of time steps, *steps* - number of time steps to calculate, *oType* - must be 'PUT' or 'CALL', *earlyExercise* - true for American, false for European




```
function oPrice = binPriceJR(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the price of a vanilla European or American
% Put or Call option using an equal-probability (Jarrow-Rudd) binomial tree.

% Author: Phil Goddard (phil@goddardconsulting.ca)

% Calculate the equal probability model parameters
p = 0.5;
u = exp((r-sig*sig/2)*dt + sig*sqrt(dt));
d = exp((r-sig*sig/2)*dt - sig*sqrt(dt));

% Loop over each node and calculate the JR underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end

% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
        valueTree(:,end) = max(priceTree(:,end)-X,0);
end
```

SA

```
28
29      % Loop backwards to get values at the earlier times
30      steps = size(priceTree,2)-1;
31      for idx = steps:-1:1
32          valueTree(1:idx,idx) = ...
33              exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
34              + (1-p)*valueTree(2:idx+1,idx+1));
35          if earlyExercise
36              switch oType
37                  case 'PUT'
38                      valueTree(1:idx,idx) = ...
39                          max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
40                  case 'CALL'
41                      valueTree(1:idx,idx) = ...
42                          max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
43              end
44          end
45      end
46
47      % Output the option price
48      oPrice = valueTree(1);
```

```
Command Window
>> oPrice=binPriceCRRDrift(60,50,0.05,0.2,0.01,100,'CALL',false,log(60)-log(50))

oPrice =

    1.6115

fx >>
```

Jarrow-Rudd binomial risk neutral option pricing model without Financial Toolbox

Inputs: *X* – strike, S0 - stock price, *r* - risk free interest rate, *sig* – volatility, *dt* - size of time steps, *steps* - number of time steps to calculate, *oType* - must be 'PUT' or 'CALL', *earlyExercise* - true for American, false for European

SAMPLE

EDITOR PUBLISH VIEW

```
function oPrice = binPriceJRRN(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the price of a vanilla European or American
% Put or Call option using a Jarrow-Rudd Risk Neutral binomial tree.

% Author: Phil Goddard (phil@goddardconsulting.ca)

% Calculate the JRRN model parameters
a = exp(r*dt);
u = exp((r-sig*sig/2)*dt + sig*sqrt(dt));
d = exp((r-sig*sig/2)*dt - sig*sqrt(dt));
p = (a-d)/(u-d);

% Loop over each node and calculate the JRRN underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end

% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
        valueTree(:,end) = max(priceTree(:,end)-X,0);
end

% Loop backwards to get values at the earlier times
steps = size(priceTree,2)-1;
for idx = steps:-1:1
    valueTree(1:idx,idx) = ...
        exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
        + (1-p)*valueTree(2:idx+1,idx+1));
    if earlyExercise
        switch oType
            case 'PUT'
                valueTree(1:idx,idx) = ...
                    max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
            case 'CALL'
                valueTree(1:idx,idx) = ...
                    max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
        end
    end
end

% Output the option price
oPrice = valueTree(1);
```

```
Command Window
>> oPrice=binPriceJRRN(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.6239

>>
```

Leisen-Reimer binomial option pricing model without Financial Toolbox

EDITOR PUBLISH VIEW

```
function oPrice = binPriceLR(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the price of a vanilla European or American
% Put or Call option using a Leisen-Reimer binomial tree.

% Author: Phil Goddard (phil@goddardconsulting.ca)

% Create a function handle for the required cummulative distribution
% approximation
fh = @(z,n)(0.5+sign(z).*sqrt(0.25-0.25*exp(-((z/(n+1/3+0.1/(n+1))).^2)*(n+1/6))));

% Calculate the d1 and d2 parameters of the Black-Scholes model
T = dt*steps ; % time to maturity
d1 = (log(S0/X)+(r+sig*sig/2)*T)/sig/sqrt(T);
d2 = (log(S0/X)+(r-sig*sig/2)*T)/sig/sqrt(T);

% Calculate the Leisen-Reimer model parameters
% The parameter n must be odd
if mod(steps,2) > 0
    n = steps;
else
    n = steps+1;
end
pbar = fh(d1,n);
p = fh(d2,n);
u = exp(r*dt)*pbar/p;
d = (exp(r*dt)-p*u)/(1-p);
```

```
28
29      % Loop over each node and calculate the CRR underlying price tree
30      priceTree = nan(steps+1,steps+1);
31      priceTree(1,1) = S0;
32      for idx = 2:steps+1
33          priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
34          priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
35      end
36
37      % Calculate the value at expiry
38      valueTree = nan(size(priceTree));
39      switch oType
40          case 'PUT'
41              valueTree(:,end) = max(X-priceTree(:,end),0);
42          case 'CALL'
43              valueTree(:,end) = max(priceTree(:,end)-X,0);
44      end
45
46      % Loop backwards to get values at the earlier times
47      steps = size(priceTree,2)-1;
48      for idx = steps:-1:1
49          valueTree(1:idx,idx) = ...
50              exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
51              + (1-p)*valueTree(2:idx+1,idx+1));
52          if earlyExercise
53              switch oType
54                  case 'PUT'
55                      valueTree(1:idx,idx) = ...
56                          max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
57                  case 'CALL'
58                      valueTree(1:idx,idx) = ...
59                          max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
60              end
61          end
62      end
63
64      % Output the option price
65      oPrice = valueTree(1);
```

Command Window

```
>> oPrice = binPriceLR(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.5931

fx >>
```

Tian binomial option pricing model without Financial Toolbox

EDITOR PUBLISH VIEW

```
function oPrice = binPriceTian(X,S0,r,sig,dt,steps,oType,earlyExercise)
% Function to calculate the price of a vanilla European or American
% Put or Call option using a Tian binomial tree.

% Author: Phil Goddard (phil@goddardconsulting.ca)

% Calculate the Tian model parameters
nu = exp(sig*sig*dt);
a = exp(r*dt);
u = 0.5*a*nu*(nu+1+sqrt(nu*nu+2*nu-3));
d = 0.5*a*nu*(nu+1-sqrt(nu*nu+2*nu-3));
p = (a-d)/(u-d);

% Loop over each node and calculate the Tian underlying price tree
priceTree = nan(steps+1,steps+1);
priceTree(1,1) = S0;
for idx = 2:steps+1
    priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
    priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
end

% Calculate the value at expiry
valueTree = nan(size(priceTree));
switch oType
    case 'PUT'
        valueTree(:,end) = max(X-priceTree(:,end),0);
    case 'CALL'
```

SA

```
28 -            valueTree(:,end) = max(priceTree(:,end)-X,0);
29 -        end
30
31          % Loop backwards to get values at the earlier times
32 -        steps = size(priceTree,2)-1;
33 -        for idx = steps:-1:1
34 -            valueTree(1:idx,idx) = ...
35                  exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
36                  + (1-p)*valueTree(2:idx+1,idx+1));
37 -            if earlyExercise
38 -                switch oType
39 -                    case 'PUT'
40 -                        valueTree(1:idx,idx) = ...
41                              max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
42 -                    case 'CALL'
43 -                        valueTree(1:idx,idx) = ...
44                              max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
45 -                end
46 -            end
47 -        end
48
49          % Output the option price
50 -        oPrice = valueTree(1);
```

```
Command Window
>> oPrice=binPriceTian(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.6293

fx >>
```

Portfolios models

Global Minimum Variance Efficient Portfolios (GMVP) without Toolboxes using langrangien multiplier

This is based on the formal solution see during the theoretical training (Langrang Multiplier):

EDITOR PUBLISH VIEW

```
clear all;close all;clc
returns = xlsread('E:\MarkowitzData.xls');

muAsset=mean(returns);
muAsset=muAsset';
sigmaAsset=cov(returns);
N = length(muAsset);
% Calculation of parameters sigma*sigma = A B C D
A = ones(N,1)'*inv(sigmaAsset)*muAsset;
B = muAsset'*inv(sigmaAsset)*muAsset;
C = ones(N,1)'*inv(sigmaAsset)*ones(N,1);
D = B*C - A^2;
H = (B*inv(sigmaAsset)*ones(N,1) - A*inv(sigmaAsset)*muAsset)/D;
G = (C*inv(sigmaAsset)*muAsset - A*inv(sigmaAsset)*ones(N,1))/D;
%Minimum variance portfolio
HH = H'*sigmaAsset*H;
GG = G'*sigmaAsset*G;
gh = H'*sigmaAsset*G;

mumin = - gh/GG;
sdmin = sqrt( HH * ( 1 - gh^2/(HH*GG)) );
% Any portfolio is on the efficient frontier can be expressed as W = Ge + H
wmin = (G*mumin + H)';
disp('-------------Minimum variance Portfolio ---------');
disp('    mumin      sdmin        ');
disp([mumin,sdmin]);
disp('                Weights             ');
disp(wmin);

muPortfolio = linspace(min(muAsset),max(muAsset),50);    %muP grid
```

```
muPortfolio = linspace(min(muAsset),max(muAsset),50);    %muP grid
sigmaPortfolio = zeros(1,50); %Storage

for i=1:50 ;
    wPortfolio = H + muPortfolio(i)*G ;% weight of ith portfolio
    sigmaPortfolio(i) = sqrt(wPortfolio'*sigmaAsset*wPortfolio) ;%risk
end ;

fsize = 16 ;
ind = (muPortfolio > mumin);    %Indicates efficient horizon
ind2 = (muPortfolio < mumin);   %Indicates locus below efficient horizon

%Create plot - efficient horizon is shown as a solid curve
%-  the inefficient part of the locus is dashed
p1 = plot(sigmaPortfolio(ind),muPortfolio(ind),'-',...
    sigmaPortfolio(ind2),muPortfolio(ind2),'--' , ...
    sdmin,mumin,'.') ;

%Change line widths, marker sizes, and colors for better appearance
set(p1(1:2),'linewidth',4) ;
set(p1(1:2),'color','blue') ;
set(p1(3),'markersize',40) ;
set(p1(3),'color','red') ;
xlabel('Risk(standard deviation)','fontsize',fsize) ;
ylabel('Expected return(mean)','fontsize',fsize) ;
grid on;
```

```
Command Window
   -------------Minimum variance Portfolio ---------
       mumin      sdmin
       0.2230     0.0462

                  Weights
       0.0194     1.0134    -0.0328

fx >>
```

Global Minimum Variance Efficient Portfolios (GMVP) using only Optimization Toolbox

For this first classical example we will us the following file:

MeanVariancePortoflioReturn.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
ABC     -.7022472       2.758133        -4.335857       6.978417        5.312709
ANZBY   -.9270704       5.926388        -6.124853       5.834379        -3.556607
APD     1.621528        6.297742        1.069946        -7.20493        2.55406 .11622
APU     1.148649        -2.004008       -.613497        9.94513 -1.840302       4.3215
ASH     5.134116        6.358381        3.335832        -.3264418       1.564774
ATO     1.279708        -.4693141       -2.067465       -2.592592       7.566544
BDX     -.2640845       5.684025        -2.421914       .1711743        -1.828433
BEN     -2.569993       3.43354 -2.194045       .0436999        5.037856        6.7091
BJS     3.266008        3.953392        3.843075        -6.013878       3.281379
CFFN    -.0277778       1.333704        -5.017823       -1.991917       -2.444775
CHFN    -8.554438       -9.47974        -7.792208       -5.543902       10.40609
COL     8.77282 7.342658        3.344191        -3.593192       7.650391        -3.462
CUB     -9.054647       -17.80702       1.067236        -7.127772       .0568527
DMF     -4.7    -.8394544       -5.502645       .3359463        1.897323        1.5334
EMR     -4.079886       -1.368233       -2.08082        -3.480135       6.062545
FICO    -5.806979       -2.170767       1.893491        -4.529617       4.166667
GLAD    1.434599        2.329451        -13.73984       3.675778        3.272724
HP      11.3396 5.646438        -.8491508       -3.173804       7.908428        13.114
IGT     -8.958697       -2.683706       -12.47538       .8627157        4.797324
IMKTA   5.972558        -.9139376       2.382782        -2.927928       -2.242843
ITYBY   -3.710407       1.842105        -1.882614       9.123401        -6.154114
JJSF    -1.590538       -2.154994       -.8047438       4.526046        .8169947
LDR     2.275711        2.931108        -1.184785       -4.606647       7.717748
LEO     2.256532        -.8130081       -4.332553       2.570379        2.386634
LG      -2.728732       3.564356        -6.9471 -6.267123       9.243696        6.2207
MATW    -5.679348       .201671 -5.807936       8.333333        4.226547        5.3257
MMS     -3.341902       12.56649        -1.092735       -8.330845       11.69381
NABZY   2.408564        -.5226481       -3.896672       4.236902        3.933569
NASB    0       5.155155        -5.759162       -1.237374       -.3068226       12.464
NFG     -.4940014       .4255319        .9533898        -4.756908       2.827764
NJR     1.176878        1.265823        -1.959459       -.3790489       4.012451
NRGY    12      1.242236        -.7055215       1.173926        -4.366413       -.0957
PBNY    -3.33586        .8627451        -4.821151       -13.48039       10.57602
PIM     .9146342        .6042296        -4.054054       -.9389671       1.105847
QCOM    -12.16981       -3.195489       1.608877        -4.750205       6.82144 -11.43
QQQQ    -6.312625       -.4812834       -1.746373       -4.347826       8.862212
RDK     -2.720148       11.84834        -1.90678        -2.980561       6.990208
RJF     .62954  -2.26179        -.5416051       -10.9901        -.2224664       4.9609
RKT     -8.509234       2.883922        -6.797477       -23.68421       13.39902
```

EDITOR PUBLISH VIEW

```
1   %Authors: Ludwig. B. Chincarini / Daewhan Kim
2   clear all;close all;clc
3   %we read the ticker list dand
4   %ticker list is the first column
5   %we skip all the other columns by using (%*[^\n])
6   ticker=textread('C:\tmp\MeanVariancePortfolioReturn.txt','%s%*[^\n]');
```

```
>> ticker

ticker =

    'ABC'
    'ANZBY'
    'APD'
    'APU'
    'ASH'
    'ATO'
    'BDX'
    'BEN'
    'BJS'
    'CFFN'
    'CHFN'
    'COL'
    'CUB'
    'DMF'
    'EMR'
```

```
1 -  clear all;close all;clc;
2    % we read the ticker list and
3    % ticker list is the first column
4    % we skip all the other columns by using (%*[^\n])
5 -  ticker=textread('data_Ch9_return.txt','%s%*[^\n]');
6    % we read returns in the file where the delimiter is the Tab (\t)
7    % we read from the first row (0) and from the second column (1)
8 -  ret = dlmread('data_Ch9_return.txt', '\t', 0, 1);
```

```
>> ret

ret =

  Columns 1 through 8

   -0.7022    2.7581   -4.3359    6.9784    5.3127    7.0881    3.8760    3.9610
   -0.9271    5.9264   -6.1249    5.8344   -3.5566    1.6595   -1.6324    2.8887
    1.6215    6.2977    1.0699   -7.2049    2.5541    0.1162   -0.8955   -7.2958
    1.1486   -2.0040   -0.6135    9.9451   -1.8403    4.3216    1.8581   -1.4952
    5.1341    6.3584    3.3358   -0.3264    1.5648    5.2311    4.6136   -1.0740
    1.2797   -0.4693   -2.0675   -2.5926    7.5665    1.8028    1.2500    1.4746
   -0.2641    5.6840   -2.4219    0.1712   -1.8284   -8.6684    5.5270   -4.9485
   -2.5700    3.4335   -2.1940    0.0437    5.0379    6.7092    4.9883   -0.4702
    3.2660    3.9534    3.8431   -6.0139    3.2814    4.2097   16.2348    3.4098
   -0.0278    1.3337   -5.0178   -1.9919   -2.4448    4.1063    4.4084   -6.2222
   -8.5544   -9.4797   -7.7922   -5.5439   10.4061    0.4023    0.1717   -1.9429
    8.7728    7.3427    3.3442   -3.5932    7.6504   -3.4622    2.3490   -1.3729
   -9.0546  -17.8070    1.0672   -7.1278    0.0569    0.7955    9.7520   -5.2388
```

Workspace

Name	Value	Min
ret	<50x60 double>	-57.7...
ticker	<50x1 cell>	

EDITOR PUBLISH VIEW

```
1 -  clear all;close all;clc;
2    % we read the ticker list and
3    % ticker list is the first column
4    % we skip all the other columns by using (%*[^\n])
5 -  ticker=textread('data_Ch9_return.txt','%s%*[^\n]');
6    % we read returns in the file where the delimiter is the Tab (\t)
7    % we read from the first row (0) and from the second column (1)
8 -  ret=dlmread('data_Ch9_return.txt', '\t', 0, 1);
9    % mean vector of each ticker (annualized) and because each
10   % ticker is horizontal in the text file we take the transposition
11   % of the vector ret
12 -  mu=mean(ret')'*12;
13   % we calculate the variance-covariance matrix (annualized)
14   % and once again we must not forget to take the transposition
15   % because there are 50 tickers, the matrix will be a 50x50
16 -  covar=cov(ret')*12;
```

Command Window

```
>> mu(1:5);
>> covar

covar =

   1.0e+03 *

  Columns 1 through 8

    0.4523    0.2493    0.1892    0.0708    0.4811    0.1065    0.1837    0.3385
    0.2493    1.0941    0.3568    0.0935    1.2169    0.1350    0.1484    0.5929
    0.1892    0.3568    0.6176    0.1618    1.3043    0.1107    0.1731    0.4439
    0.0708    0.0935    0.1618    0.3724    0.4626    0.1075    0.1111    0.1184
    0.4811    1.2169    1.3043    0.4626    5.6584    0.4254    0.2544    1.1482
    0.1065    0.1350    0.1107    0.1075    0.4254    0.2443    0.0485    0.1213
    0.1837    0.1484    0.1731    0.1111    0.2544    0.0485    0.4345    0.1431
    0.3385    0.5929    0.4439    0.1184    1.1482    0.1213    0.1431    0.8210
    0.2006    0.4046    0.4963    0.1149    1.7222    0.1654    0.1385    0.3990
    0.0338    0.0354    0.0334    0.0480    0.3126    0.0006    0.0415    0.1239
```

Workspace

Name	Value	Min
ans	[14.3968;10.1940;9.8...	7.5220
covar	<50x50 double>	-500....
mu	<50x1 double>	-21.0...
ret	<50x60 double>	-57.7...
ticker	<50x1 cell>	

SAMPLE

EDITOR PUBLISH VIEW

```
1  -    clear all;close all;clc;
2       % we read the ticker list and
3       % ticker list is the first column
4       % we skip all the other columns by using (%*[^\n])
5  -    ticker=textread('data_Ch9_return.txt','%s%*[^\n]');
6       % we read returns in the file where the delimiter is the Tab (\t)
7       % we read from the first row (0) and from the second column (1)
8  -    ret=dlmread('data_Ch9_return.txt', '\t', 0, 1);
9       % mean vector of each ticker (annualized) and because each
10      % ticker is horizontal in the text file we take the transposition
11      % of the vector ret
12 -    mu=mean(ret')'*12;
13      % we calculate the variance-covariance matrix (annualized)
14      % and once again we must not forget to take the transposition
15      % because there are 50 tickers, the matrix will be a 50x50
16 -    covar=cov(ret')*12;
17      % determine the number of assets
18 -    n=size(mu,1);
19      % determine min and max of mean returns
20      % because we can't have a mean return better or worst than
21      % the min or max return
22 -    mu_min=min(mu);mu_max=max(mu);
```

Command Window

```
>> n

n =

     50

>> mu_max

mu_max =

   31.4589

>> mu_min

mu_min =

  -21.0550

fx >>
```

EDITOR PUBLISH VIEW

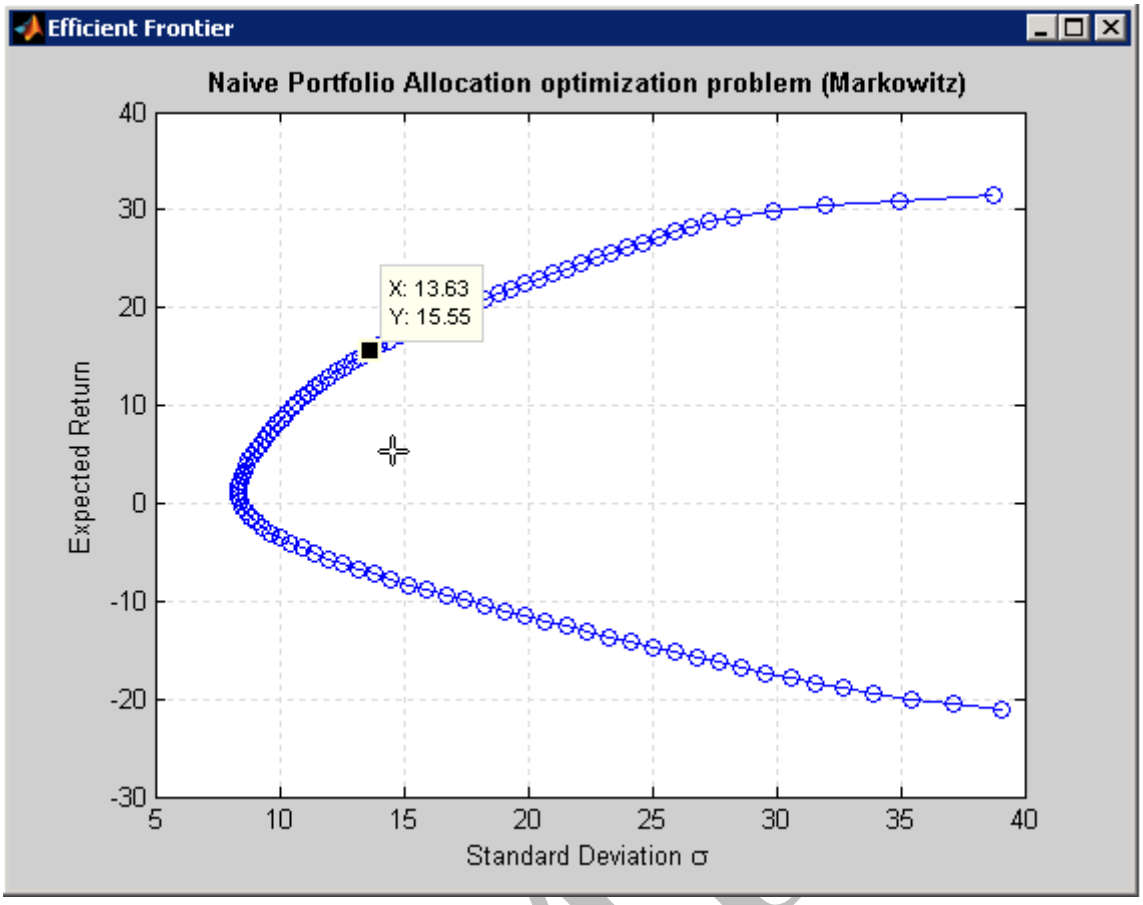
```
clear all;close all;clc;
% we read the ticker list and
% ticker list is the first column
% we skip all the other columns by using (%*[^\n])
ticker=textread('data_Ch9_return.txt','%s%*[^\n]');
% we read returns in the file where the delimiter is the Tab (\t)
% we read from the first row (0) and from the second column (1)
ret=dlmread('data_Ch9_return.txt', '\t', 0, 1);
% mean vector of each ticker (annualized) and because each
% ticker is horizontal in the text file we take the transposition
% of the vector ret
mu=mean(ret')'*12;
% we calculate the variance-covariance matrix (annualized)
% and once again we must not forget to take the transposition
% because there are 50 tickers, the matrix will be a 50x50
covar=cov(ret')*12;
% determine the number of assets
n=size(mu,1);
% determine min and max of mean returns
% because we can't have a mean return better or worst than
% the min or max return
mu_min=min(mu);mu_max=max(mu);
% we will decide the optimize 100 portolios with return
% between mu_min and mu_max by constant step
m_step = (mu_max - mu_min) / 99;
m_target = (mu_min:m_step:mu_max)';
%because the purpose is to found the optimal combination of weight of
%tickers first the lower bound of each ticker will be build with
lb=zeros(n,1);
% linear constraints (matrix of two rows and 50 columns)
% the first row contains fifty 1 to say we want to include the 50 tickers
% in the optimization problem, the second row contains the mean return
% as coefficient of each ticker for which we need to find the weight
Aeq = [ones(1,n);mu'];
% initialize matrices to store weights and standard deviation
% for the 100 hundred optimisations and 50 (n) tickers we will need
% a 100x50 matrix
w_all = zeros(100,n);
% we will also need a vector containing the portfolio variance
s_all = zeros(100,1);
```

EDITOR PUBLISH VIEW

```
41      % we define the settings for the optimization function of the
42      % Optimization Toolbox
43 -    opts = optimoptions('quadprog','Algorithm','interior-point-convex',...
44          'Display','off');
45
46      % we loop over target means
47 -    for i = 1:100
48          % linear constraints
49          % first coefficient is the sum of weights, second is the target return
50 -        Beq = [1;m_target(i)];
51          % quadratic programming (maximization by default)
52 -        w_i = quadprog(covar,[],[],[],Aeq,Beq,lb,[],[],opts);
53          % we compute the standard deviation of the optimized portfolio
54          % as we prooved it in the Financial Mathematics training
55          % (thus this stdev includes non-independance of tickers)
56 -        s_i = sqrt(w_i'*covar*w_i);
57          % collect weights and standard deviation
58 -        w_all(i,:) = w_i;
59 -        s_all(i) = s_i;
60 -    end
```

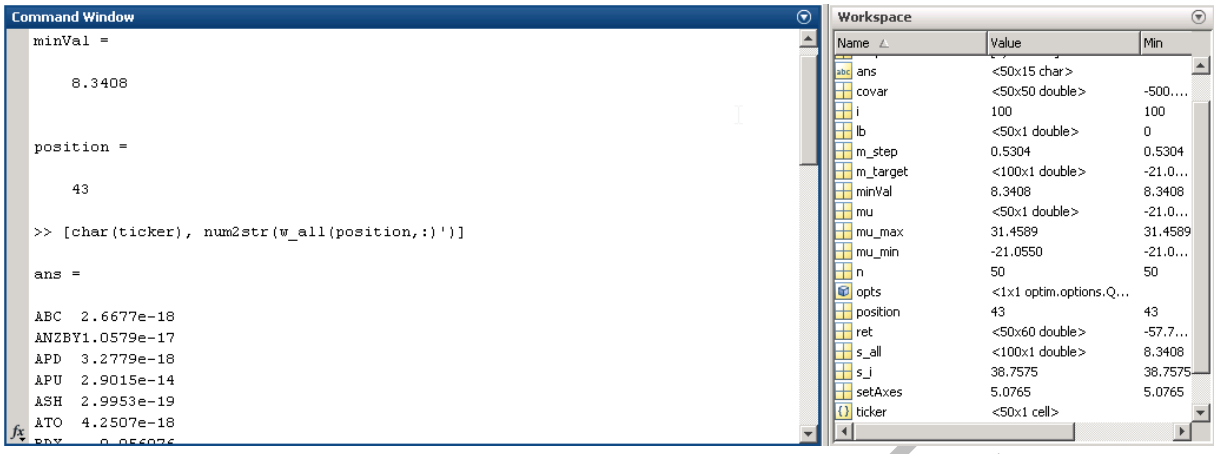
EDITOR PUBLISH VIEW

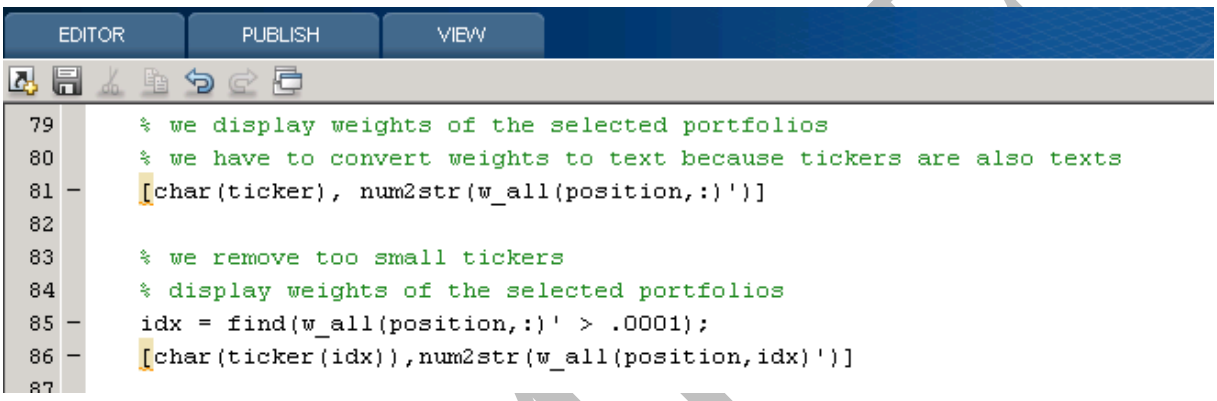
```
59 -        s_all(i) = s_i;
60 -    end
61
62      % plot the mean-variance efficient frontier
63 -    figure
64 -    set(gcf,'Menubar','none','Name','Efficient Frontier','NumberTitle','off');
65 -    plot(s_all,m_target,'-ob')
66 -    xlabel('Standard Deviation \sigma');ylabel('Expected Return');
67 -    title('Naive Portfolio Allocation optimization problem (Markowitz)',...
68          'fontweigh','bold');
69 -    grid on;
70 -    set(gca,'Xcolor',[0.85 0.85 0.85]);
71 -    set(gca,'Ycolor',[0.85 0.85 0.85]);
72 -    setAxes=copyobj(gca,gcf);
73 -    set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
74 -    datacursormode on;
```

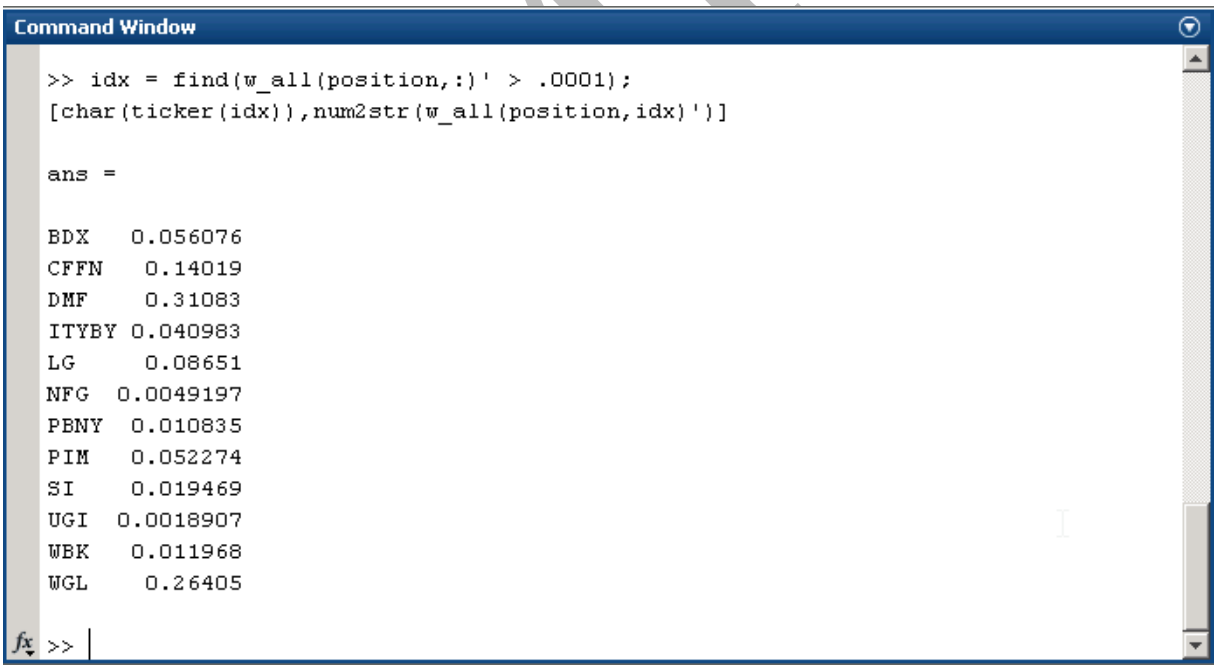



```
EDITOR  PUBLISH  VIEW

76      % we search the position number of minimum variance portfolio
77 -    [minVal, position]=min(s_all)
78
79      % we display weights of the selected portfolios
80      % we have to convert weights to text because tickers are also texts
81 -    [char(ticker), num2str(w_all(position,:)')]
```


Command Window
minVal =
8.3408
position =
43
>> [char(ticker), num2str(w_all(position,:)')]
ans =
ABC 2.6677e-18
ANZBY1.0579e-17
APD 3.2779e-18
APU 2.9015e-14
ASH 2.9953e-19
ATO 4.2507e-18
Workspace
Name
Value
Min
ans <50x15 char>
covar <50x50 double> -500....
i 100 100
lb <50x1 double> 0
m_step 0.5304 0.5304
m_target <100x1 double> -21.0...
minVal 8.3408 8.3408
mu <50x1 double> -21.0...
mu_max 31.4589 31.4589
mu_min -21.0550 -21.0...
n 50 50
opts <1x1 optim.options.Q...
position 43 43
ret <50x60 double> -57.7...
s_all <100x1 double> 8.3408
s_i 38.7575 38.7575
setAxes 5.0765 5.0765
ticker <50x1 cell>


EDITOR
PUBLISH
VIEW
79 % we display weights of the selected portfolios
80 % we have to convert weights to text because tickers are also texts
81 [char(ticker), num2str(w_all(position,:)')]
82
83 % we remove too small tickers
84 % display weights of the selected portfolios
85 idx = find(w_all(position,:)' > .0001);
86 [char(ticker(idx)),num2str(w_all(position,idx)')]


Command Window
>> idx = find(w_all(position,:)' > .0001);
[char(ticker(idx)),num2str(w_all(position,idx)')]
ans =
BDX 0.056076
CFFN 0.14019
DMF 0.31083
ITYBY 0.040983
LG 0.08651
NFG 0.0049197
PBNY 0.010835
PIM 0.052274
SI 0.019469
UGI 0.0018907
WBK 0.011968
WGL 0.26405
>>

Portfolio Performance

EDITOR PUBLISH VIEW

```
clear all;close all;clc
ra=[-0.15,0.05,-0.43,0.79,0.32];
rb=[0.29,0.18,0.24,0.25,0.17];
rc=[0.38,0.63,0.46,0.36,-0.57];
PfM=[0.45,0.61,0.54,0.29,0.12];
w=[-0.055,1.079,-0.024];
Rf=repmat(0,1,length(PfM));
muAsset=mean([ra',rb',rc']);
sigmaAsset=std([ra',rb',rc'])';
Returns=[ra',rb',rc',PfM'];
muPortfolio = muAsset*w';

[m,n]=size(Returns);
%compute the excess return
exreturn = zeros(m,n);
for i = 1:m
    exreturn(i,:)=Returns(i,:)- Rf(i);
end

```

SAMP

```
%regression with each risky asset on the portfolio, note that the
%last columns of the exreturn matrix is  market premium data
for i= 1:n-1
    CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Covariance matrix
    % beta = cov(x,y)/var(x);
    Beta(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
    % Alpha = return - (rf + beta(E(rm)-rf))
    Alpha(i)= mean(exreturn(:,i))-(mean(Rf)+ Beta(i)*mean(exreturn(:,n)));
    % rt = rf + beta(E(rm)-rf)
    Rt(i) = mean(Rf) +  Beta(i)*mean(exreturn(:,n));
    %Calculating the Sharpe ratios for individual stocks
    Sharpe(i) = mean(exreturn(:,i))/sigmaAsset(i);
    %Calculating the Treynor ratios for individual stocks
    Treynor(i) = mean(exreturn(:,i))/Beta(i);
end

%Asset performance
Alpha
Beta
Rt
Treynor
Sharpe
%Portfolio performance
pA = muPortfolio - Rt*w'
pB = Beta*w'
pR = Rt*w'
pT = Treynor*w'
pS = Sharpe*w'
```

SA

```
Command Window
Alpha =
    0.7186    0.2041   -0.6029
Beta =
   -1.4990    0.0545    2.1265
Rt =
   -0.6026    0.0219    0.8549
Treynor =
   -0.0774    4.1463    0.1185
Sharpe =
    0.2489    4.4931    0.5343
pA =
    0.1952
pB =
    0.0902
pR =
    0.0363
pT =
    4.4753
pS =
    4.8216
fx >>
```

Michaud Optimal Portfolio (bootstrap/resample efficiency)

EDITOR PUBLISH VIEW

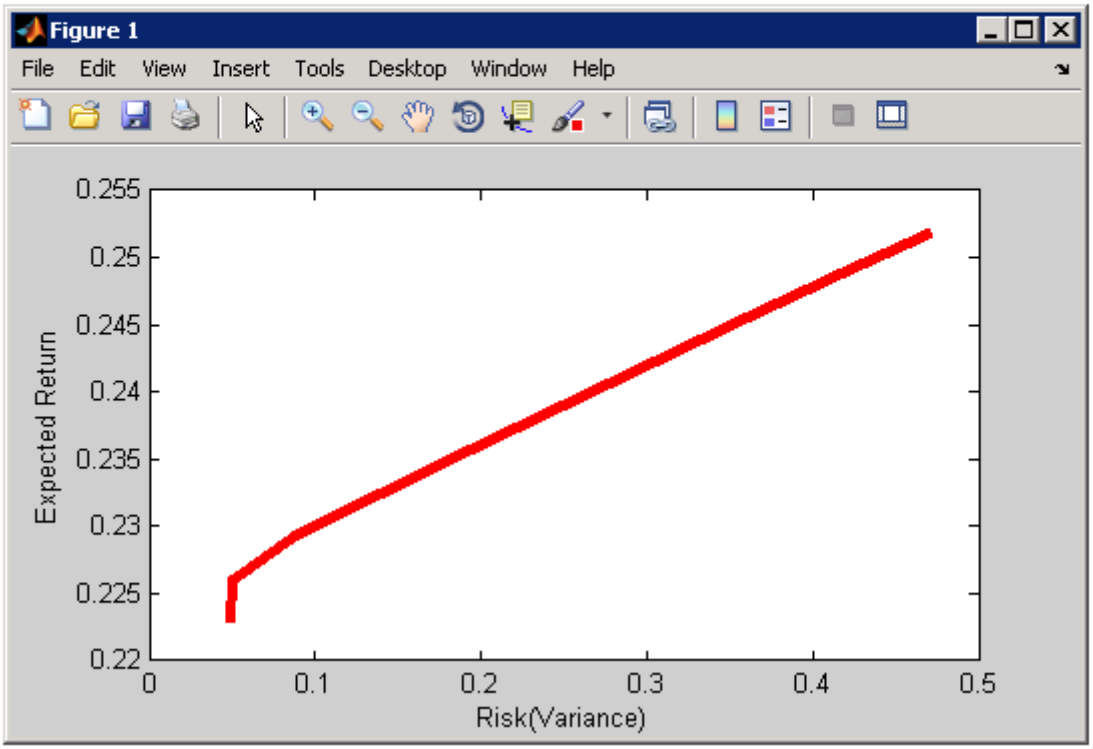
```
1   %Michaud first step is standard Markowitz Optimization
2   %unconstrained in this case)
3   clear all;close all;clc
4   T1=[-0.15,0.05,-0.43,0.79,0.32];
5   T2=[0.29,0.18,0.24,0.25,0.17];
6   T3=[0.38,0.63,0.46,0.36,-0.57];
7   listed_Asset = {'T1','T2','T3'};
8   Data=[T1' T2' T3'];
9   expReturns =mean(Data);
10  m_Cov = cov(Data);
11  stdDev = std(Data);
12  [nrow,NumAssets]= size(Data);
13  NumSim = 10; % Number of samples
14  %Markowitz Optimization
15  [PortRisk, PortReturn, PortWts] = portopt(expReturns,m_Cov,NumSim);
16  plot(PortRisk,PortReturn,'r','linewidth',4);
17  ylabel('Expected Return');xlabel('Risk(Variance)')
18  [RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
19      = portalloc(PortRisk, PortReturn, PortWts,0.001,0.001);
20  Simple_MV_Optimization = RiskyWts;
21  Simple_MV_Optimization
```

```
Command Window

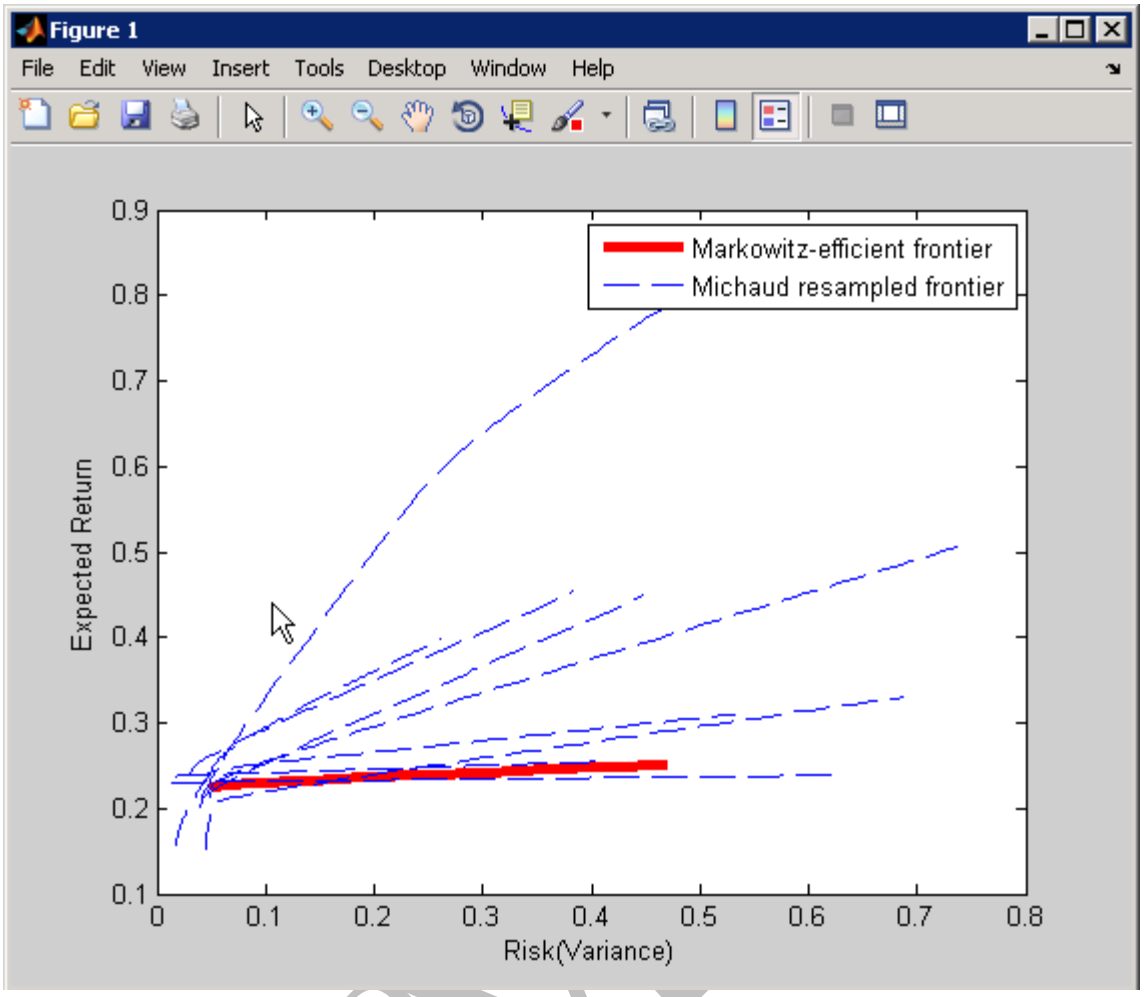
Simple_MV_Optimization =

    0.0163    0.9837         0

>>
```




```
22
23      %Second step is just a resampling routine (the core of Michaud model)
24      %to manage the estimate of covariance matrix and all other stuff
25      %if the dataset is too small the script could stop...
26 -    i=1;
27 -    hold on;
28 -    for j=1:NumSim
29          %Resample from the past returns with normal distribution
30 -        returnsim=ones(nrow,1)*expReturns+randn(nrow,NumAssets).*(ones(nrow,1)*stdDev);
31 -        meanSim=mean(returnsim);% sample mean
32 -        covSim=cov(returnsim);% sample covariance
33 -        [PortRisk, PortReturn, PortWts] = portopt(meanSim,covSim,100);
34 -        plot(PortRisk,PortReturn,'--');% plot (i)th sample frontier
35 -        [RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
36              = portalloc(PortRisk, PortReturn, PortWts, .02,0.02);
37 -        weight(:,i)= RiskyWts;% get (i)th sample optimisation portfolio
38 -        i = i + 1;
39 -    end
40 -    legend('Markowitz-efficient frontier','Michaud resampled frontier')
41 -    hold off
```




```
42
43 -    weight = weight';
44 -    Michaud_resampling_portfolio = mean(weight);
45 -    Michaud_resampling_portfolio
46
47      %resume all as bar chart
48 -    figure
49 -    subplot(2,1,1)
50 -    bar(Simple_MV_Optimization)
51 -    ylabel('Weigts');
52 -    set(gca,'XTickLabel',listed_Asset)
53 -    title('M-V Optimization portfolio');
54 -    subplot(2,1,2)
55 -    bar(Michaud_resampling_portfolio,'r')
56 -    ylabel('Weigts')
57 -    set(gca,'XTickLabel',listed_Asset)
58 -    title('Resampling Optimization portfolio ')
```

```
Command Window

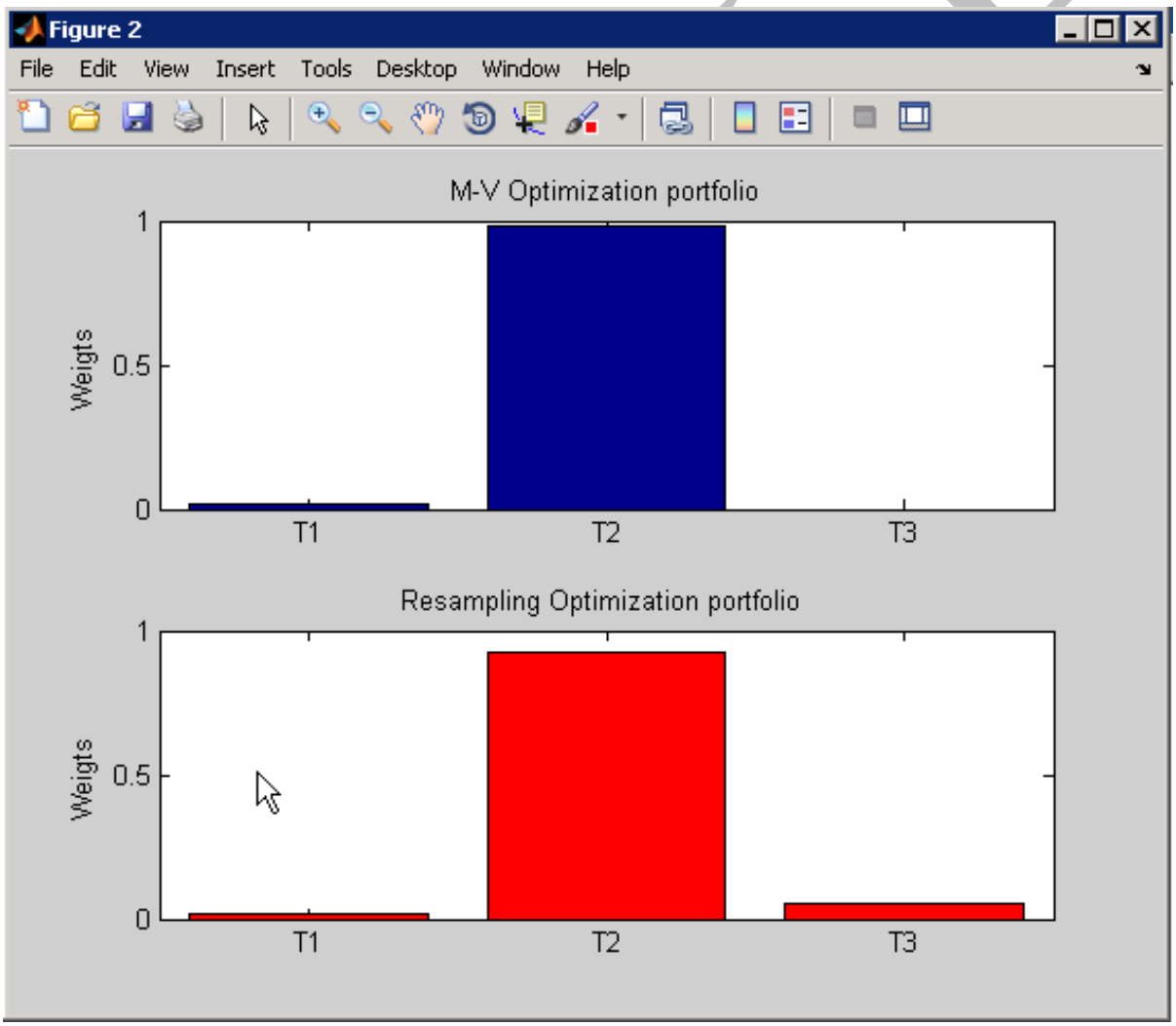
Simple_MV_Optimization =

    0.0163    0.9837         0


Michaud_resampling_portfolio =

    0.0628    0.8857    0.0516

fx >>
```



CVaR portfolio

EDITOR PUBLISH VIEW

cVaR.m

```
function result = cVaR(w,data,alpha)
% cVaR calculation for scenario returns
% w is in (n x 1) vector and r is a (T x n) vector, where
% n is the number of assets and T the number of simulations
[T,n] = size(data);
rp = sort(data*w); % sorted portfolio return
cutoff = floor(alpha*T); % cutoff (VaR)
result = mean(rp(1:cutoff,1));
end
```

EDITOR PUBLISH VIEW

cVaR_Optimization.m

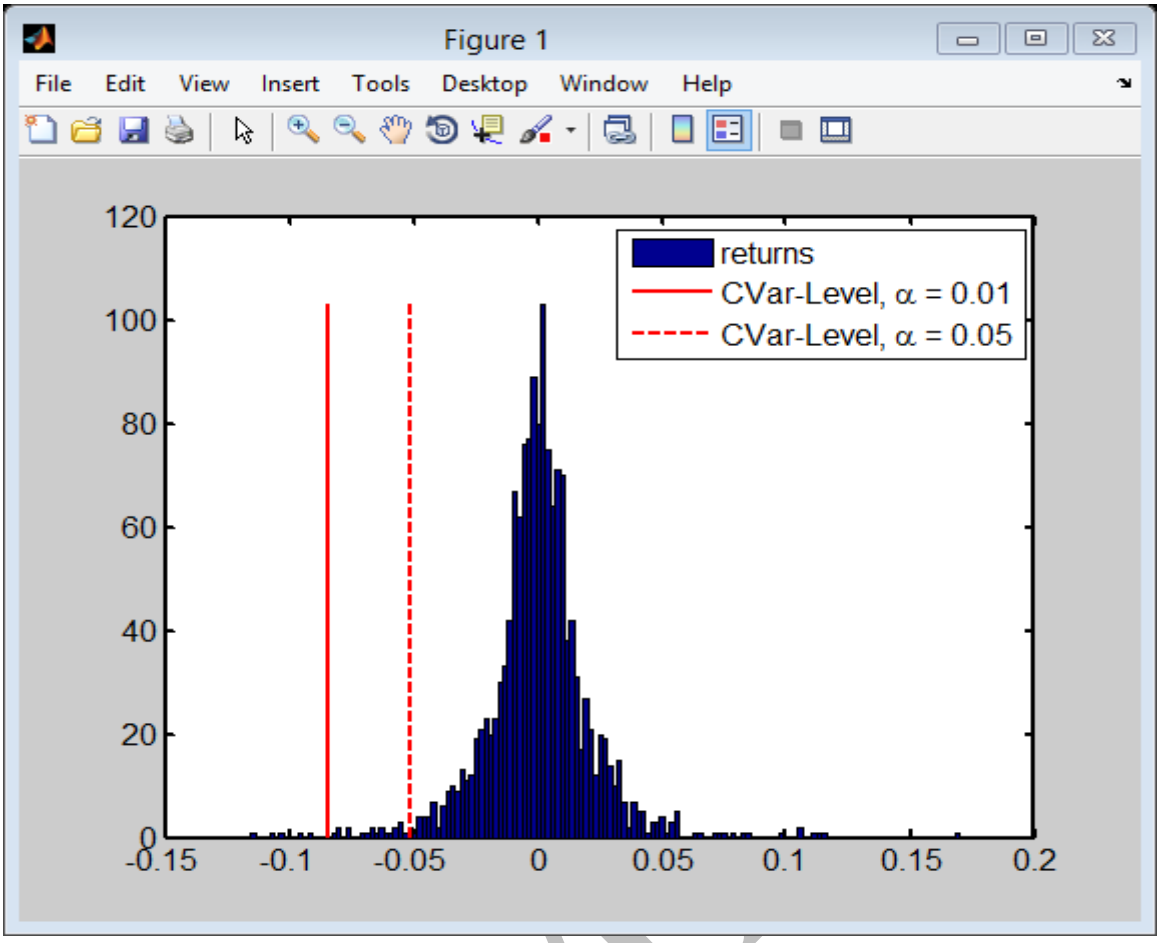
```
close all
clear all
clc
%% Loading Data
[data ,symb] = xlsread('Stocks.xlsx');
[T ,N]   = size(data);
covAsset   = cov(data);
muAsset     = mean(data); % This definition is again arbitrary to get some reasonable values
% Plotting the distribution of the equal weighted portfolio
wgtn = ones(N,1)/N; % equal weighted portfolio weights
figure(1)
prob1 = 0.01;% Alpha
CVaR1 = cVaR(wgtn,data,prob1);
prob2 = 0.05; % Alpha
CVaR2 = cVaR(wgtn,data,prob2);
hist1 = hist(data*wgtn,T/10);
hist(data*wgtn,T/10)
set(gca,'Box', 'On', 'LineWidth',1.5, 'FontSize', 12)
hold on
plot([CVaR1 CVaR1],[0 max(hist1)],'r', 'LineWidth', 1.5)
plot([CVaR2 CVaR2],[0 max(hist1)],'r--', 'LineWidth', 1.5)
legend('returns',['CVar-Level, \alpha = ' num2str(prob1)],['CVar-Level, \alpha = ' num2str(prob2)])

%% Optimization
% first, specify the target returns:
Rs = linspace(max(min(muAsset),0), max(muAsset),15);
% then the confidence level alpha
alpha=0.005;
% set the lower und upper bounds
UB=   1;
LB = -UB;
Short=0; %If Short=1, we allow for short positions. If short = 0, we don't.
% here comes the optimization loop:
clear ES1 R
```

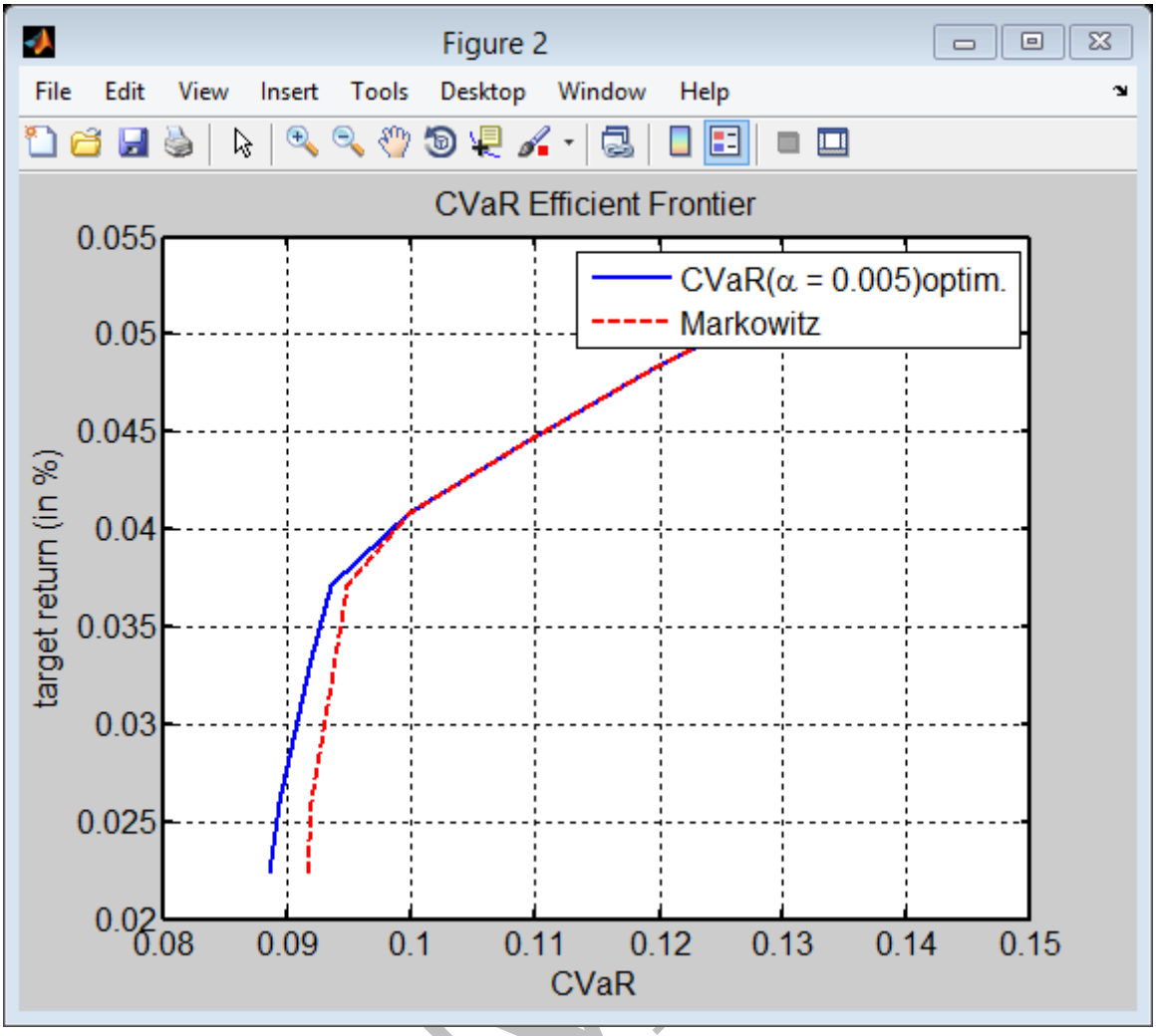
```
35      % we want to minimize -CVaR.
36 -    f=@(w) -(cVaR(w,data,alpha));
37      % and the Markowitz problem:
38 -    meanvar=@(w) w'*covAsset*w;
39      %Initial guess:
40 -    w0  = wgtn;
41 -    ES0 = cVaR(w0,data,alpha);
42 -    for i = 1:length(Rs)
43 -        R0=  Rs(i);    % target return i
44 -        A= - mean(data);
45 -        if Short==0
46 -            A=[A;-eye(N)];
47 -            A=[A; eye(N)];
48 -            b=[-R0 zeros(1,N) UB*ones(1,N)];
49 -        elseif Short==1
50 -            A=[A;-eye(N)];
51 -            A=[A; eye(N)]; %#ok<*AGROW>
52 -            b=[-R0 -LB*ones(1,N) UB*ones(1,N)];
53 -        elseif Short~=0||Short~=1
54 -            error('Set Short=1 or 0')
55 -        end
56 -        b=b';
57 -        Aeq= ones(1,N);
58 -        beq= 1;
59 -        options = optimset(optimset('fmincon'), 'Display', 'off', 'Algorithm', 'sqp' );
60      % CVaR optimization
61 -        [w,~,~,~] =fmincon(f,w0,A,b,Aeq,beq,[],[],[],options);
62      % Mean-Var optimization
63 -        [wm,~,~,~]=fmincon(meanvar,w0,A,b,Aeq,beq,[],[],[],options);
64 -        CVaR(i)      = -cVaR(w,data,alpha);
65 -        CVaRwm(i)    = -cVaR(wm,data,alpha);
66 -        SD(i)        =  std(data*w);
67 -        R(i)         =  mean(data*w);
```

SAMPLE

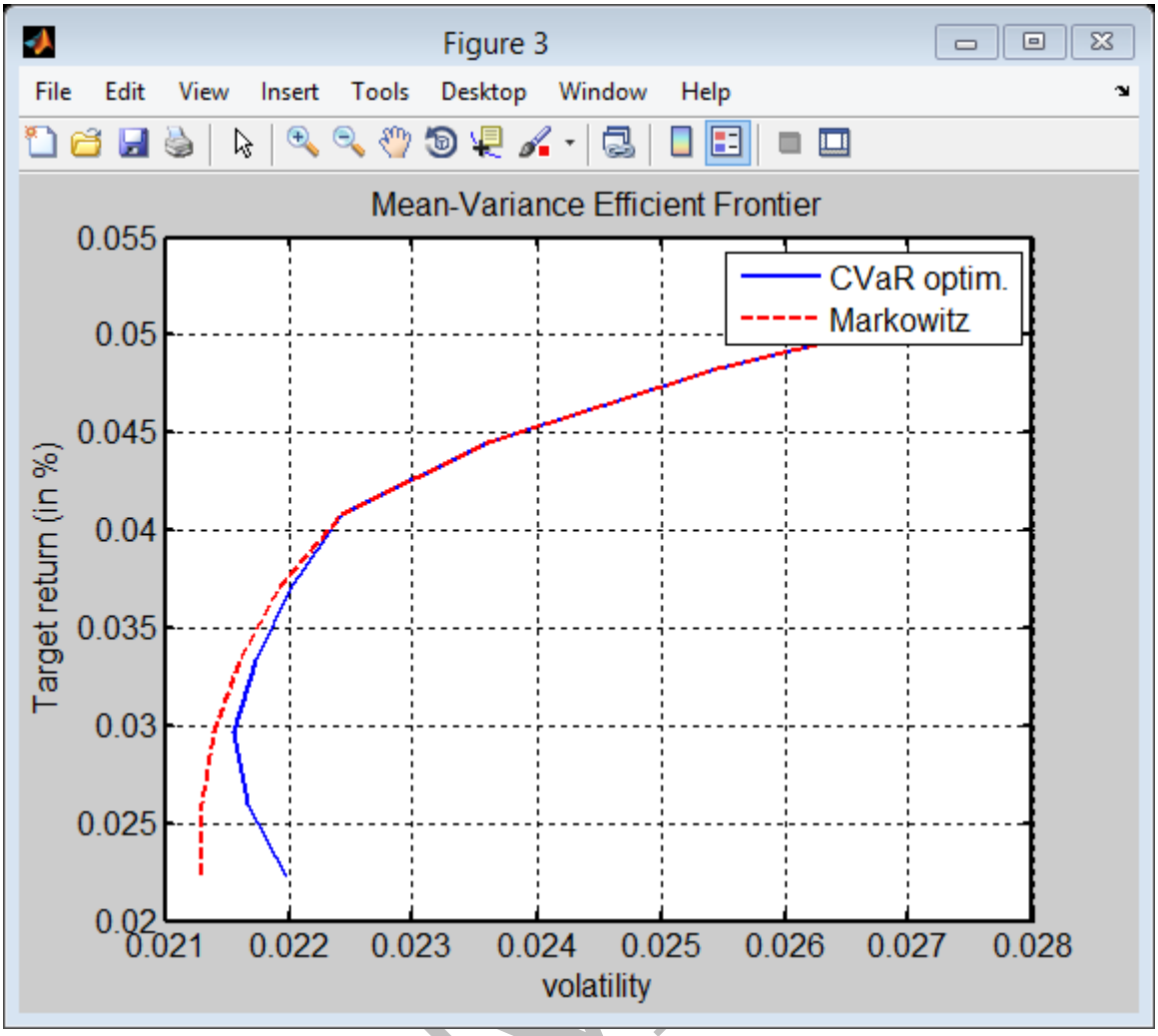
```
68 -        effSig(i)    = std(data*wm);
69 -        effMu(i)     = mean(data*w) ;
70 -        Weight(:,i) = w;
71 -        MarkoW(:,i) = wm;
72 -    end
73
74      % Plot of the CVaR efficient frontier
75 -    figure(2)
76 -    hold on
77 -    plot(CVaR,R*100,'LineWidth',1.5)
78 -    plot(CVaRwm,effMu*100,'r--','LineWidth',1.5)
79 -    set(gca,'Box','on','LineWidth', 1.5, 'FontSize',12')
80 -    grid on
81 -    ylabel('target return (in %)')
82 -    title('CVaR Efficient Frontier')
83 -    legend(['CVaR(\alpha = ' num2str(alpha) ')optim.'],'Markowitz')
84 -    xlabel('CVaR')
85
86      % Plot of the Mean-Variance efficient frontier
87 -    figure(3)
88 -    plot(SD,R*100,'LineWidth',1.5)
89 -    hold on
90 -    plot(effSig,effMu*100,'r--','LineWidth',1.5)
91 -    set(gca,'Box','on','LineWidth', 1.5, 'FontSize',12')
92 -    grid on
93 -    ylabel('Target return (in %)')
94 -    legend('CVaR optim.','Markowitz')
95 -    title('Mean-Variance Efficient Frontier')
96 -    xlabel('volatility')
97
98      % Plot of the portfolio weights
99 -    figure(4)
100 -   subplot(1,2,1)
101 -   area(Weight');
102 -   title('CVaR Portfolio Weights');
103 -   set(get(gcf,'Children'),'YLim',[0 1]);
104 -   xlabel('portfolio')
105 -   ylabel('asset weight')
106
107 -   subplot(1,2,2)
108 -   area(MarkoW');
109 -   title('Markowitz Portfolio Weights');
110 -   set(get(gcf,'Children'),'YLim',[0 1]);
111 -   xlabel('portfolio')
112 -   ylabel('asset weight')
113
```


Figure 1
File Edit View Insert Tools Desktop Window Help
returns
CVar-Level, α = 0.01
CVar-Level, α = 0.05
120
100
80
60
40
20
0
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2

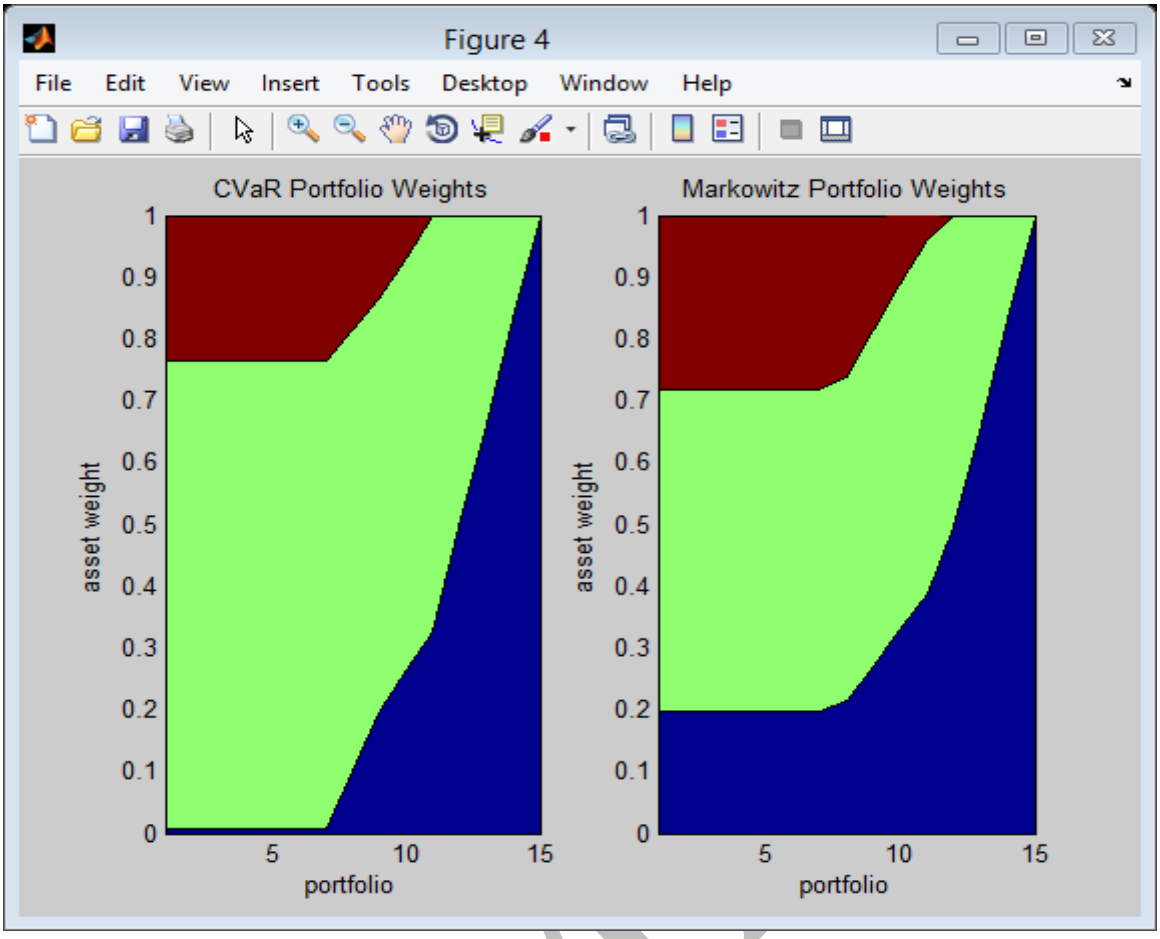
SAMPLE


Figure 2
File Edit View Insert Tools Desktop Window Help
CVaR Efficient Frontier
CVaR(α = 0.005)optim.
Markowitz
target return (in %)
0.055
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15
CVaR

SAMPLE


Figure 3
File Edit View Insert Tools Desktop Window Help
Mean-Variance Efficient Frontier
CVaR optim.
Markowitz
Target return (in %)
volatility
0.055
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.021
0.022
0.023
0.024
0.025
0.026
0.027
0.028

SAMPLE


Figure 4
File Edit View Insert Tools Desktop Window Help
CVaR Portfolio Weights
Markowitz Portfolio Weights
asset weight
portfolio
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
5 10 15

SAMPLE

Black-Littermann portfolio

EDITOR PUBLISH VIEW

```
function [er, ps, w, pw, lambda, theta] = ...
    Black_litterman(delta, weq, sigma, tau, P, Q, Omega)%#codegen
    %http://www.blacklitterman.org/Black-Litterman.pdf
    %Copyright (c) Jay Walters, blacklitterman.org, 2008.

    %Input:
    %delta - Risk tolerance from the equilibrium portfolio
    %weq - Weights of the assets in the equilibrium portfolio
    %sigma - Prior covariance matrix
    %tau - Coefficiet of uncertainty in the prior estimate of the mean (pi)
    %P - Pick matrix for the view(s)
    %Q - Vector of view returns
    %Omega - Matrix of variance of the views (diagonal)

    %Outputs
    %Er - Posterior estimate of the mean returns
    %w - Unconstrained weights computed given the posterior estimates
    %of the mean and covariance of returns.
    %lambda - A measure of the impact of each view on the posterior estimates.
    %theta  - A measure of the share of the prior and sample information
    %in the posterior precision.

    %Reverse optimize and back out the equilibrium returns
    pi = weq * sigma * delta;
    %We use tau * sigma many places so just compute it once
    ts = tau * sigma;
    %Compute posterior estimate of the mean
    er = pi' + ts * P' * inv(P * ts * P' + Omega) * (Q - P * pi');
    % We can also do it the long way to illustrate that d1 + d2 = I
```

```
30 -        d = inv(inv(ts) + P' * inv(Omega) * P);
31 -        d1 = d * inv(ts);
32 -        d2 = d * P' * inv(Omega) * P;
33 -        er2 = d1 * pi' + d2 * pinv(P) * Q;
34          %Compute posterior estimate of the uncertainty in the mean
35 -        ps = ts - ts * P' * inv(P * ts * P' + Omega) * P * ts;
36 -        posteriorSigma = sigma + ps;
37          %Compute the share of the posterior precision from prior and views,
38          %then for each individual view so we can compare it with lambda
39 -        theta=zeros(1,2+size(P,1));
40 -        theta(1,1) = (trace(inv(ts) * ps) / size(ts,1));
41 -        theta(1,2) = (trace(P'*inv(Omega)*P* ps) / size(ts,1));
42 -        for i=1:size(P,1)
43 -            theta(1,2+i) = (trace(P(i,:)'*inv(Omega(i,i))*P(i,:)* ps) / size(ts,1));
44 -        end
45          %Compute posterior weights based solely on changed covariance
46 -        w = (er' * inv(delta * posteriorSigma))';
47          % Compute posterior weights based on uncertainty in mean and covariance
48 -        pw = (pi * inv(delta * posteriorSigma))';
49          %Compute lambda value
50          %We solve for lambda from formula (17) page 7, rather than formula (18)
51          %just because it is less to type, and we've already computed w*.
52 -        lambda = pinv(P)' * (w'*(1+tau) - weq)';
53 -    end
```

Now we test the script with the values of the article of BL:

SAMPLE

EDITOR PUBLISH VIEW

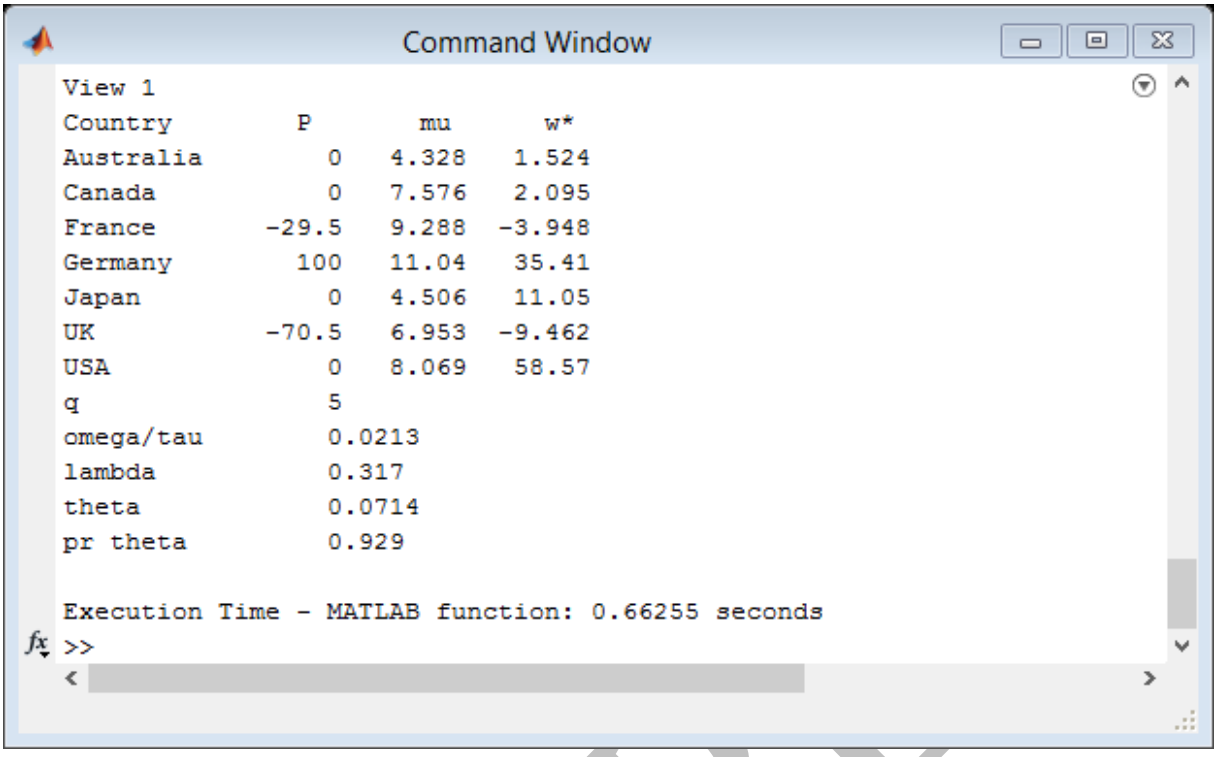
```
%Copyright 2012 The MathWorks, Inc.
%Copyright Jay Walters, blacklitterman.org, 2008.
%Take the values from He & Litterman, 1999.
weq = [0.016,0.022,0.052,0.055,0.116,0.124,0.615];
C = [ 1.000 0.488 0.478 0.515 0.439 0.512 0.491;
      0.488 1.000 0.664 0.655 0.310 0.608 0.779;
      0.478 0.664 1.000 0.861 0.355 0.783 0.668;
      0.515 0.655 0.861 1.000 0.354 0.777 0.653;
      0.439 0.310 0.355 0.354 1.000 0.405 0.306;
      0.512 0.608 0.783 0.777 0.405 1.000 0.652;
      0.491 0.779 0.668 0.653 0.306 0.652 1.000];
Sigma = [0.160 0.203 0.248 0.271 0.210 0.200 0.187];
refPi = [0.039 0.069 0.084 0.090 0.043 0.068 0.076];
assets={'Australia';'Canada   ';'France   ';'Germany  ';'Japan    ';...
    'UK       ';'USA      '};
labels={'q        ';'omega/tau';'lambda   ';'theta    ';'pr theta  '};
V = (Sigma' * Sigma) .* C;
[m,n]=size(refPi);

%Risk tolerance of the market from the paper
delta= 2.5;

%Coefficient of uncertainty in the prior estimate of the mean
tau = 0.05;

% Define view 1
% Germany will outperform the other European markets by 5%
% Market cap weight the P matrix
P1 = [0 0 -.295 1.00 0 -.705 0 ];
Q1 = [0.05];
```

SA

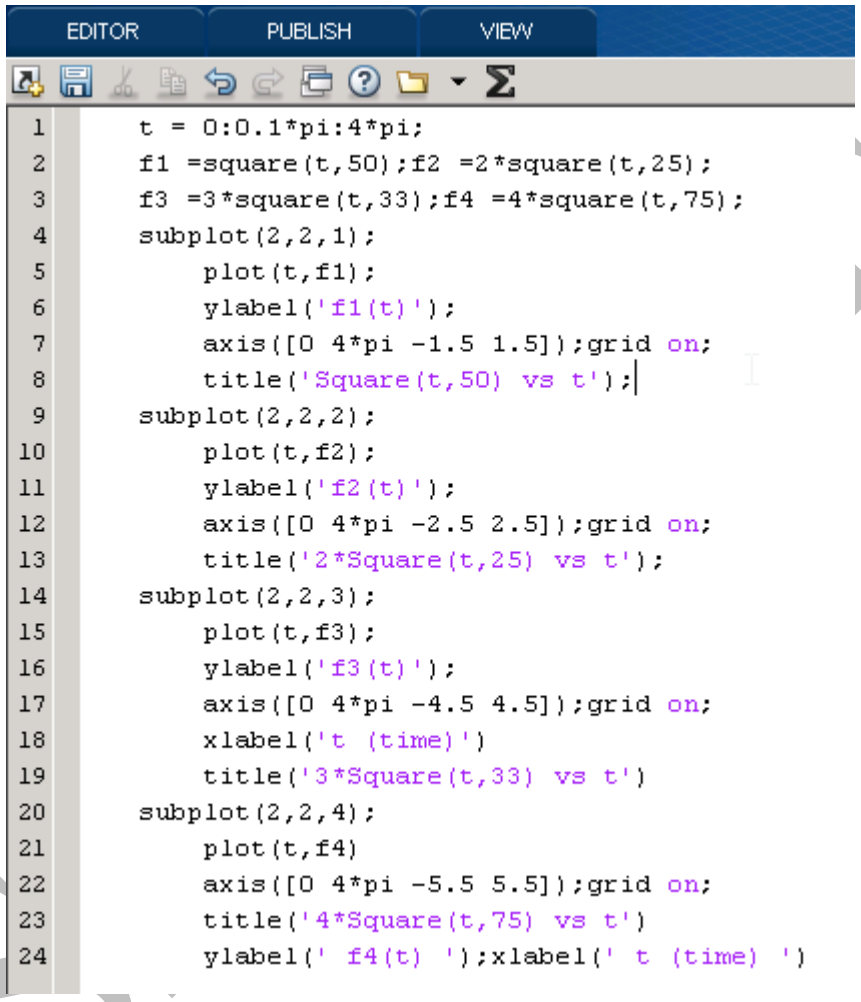

Command Window
View 1
Country P mu w*
Australia 0 4.328 1.524
Canada 0 7.576 2.095
France -29.5 9.288 -3.948
Germany 100 11.04 35.41
Japan 0 4.506 11.05
UK -70.5 6.953 -9.462
USA 0 8.069 58.57
q 5
omega/tau 0.0213
lambda 0.317
theta 0.0714
pr theta 0.929
Execution Time - MATLAB function: 0.66255 seconds
>>

SAMPLE

Signal processing (Signal processing & Communications System Toolbox)

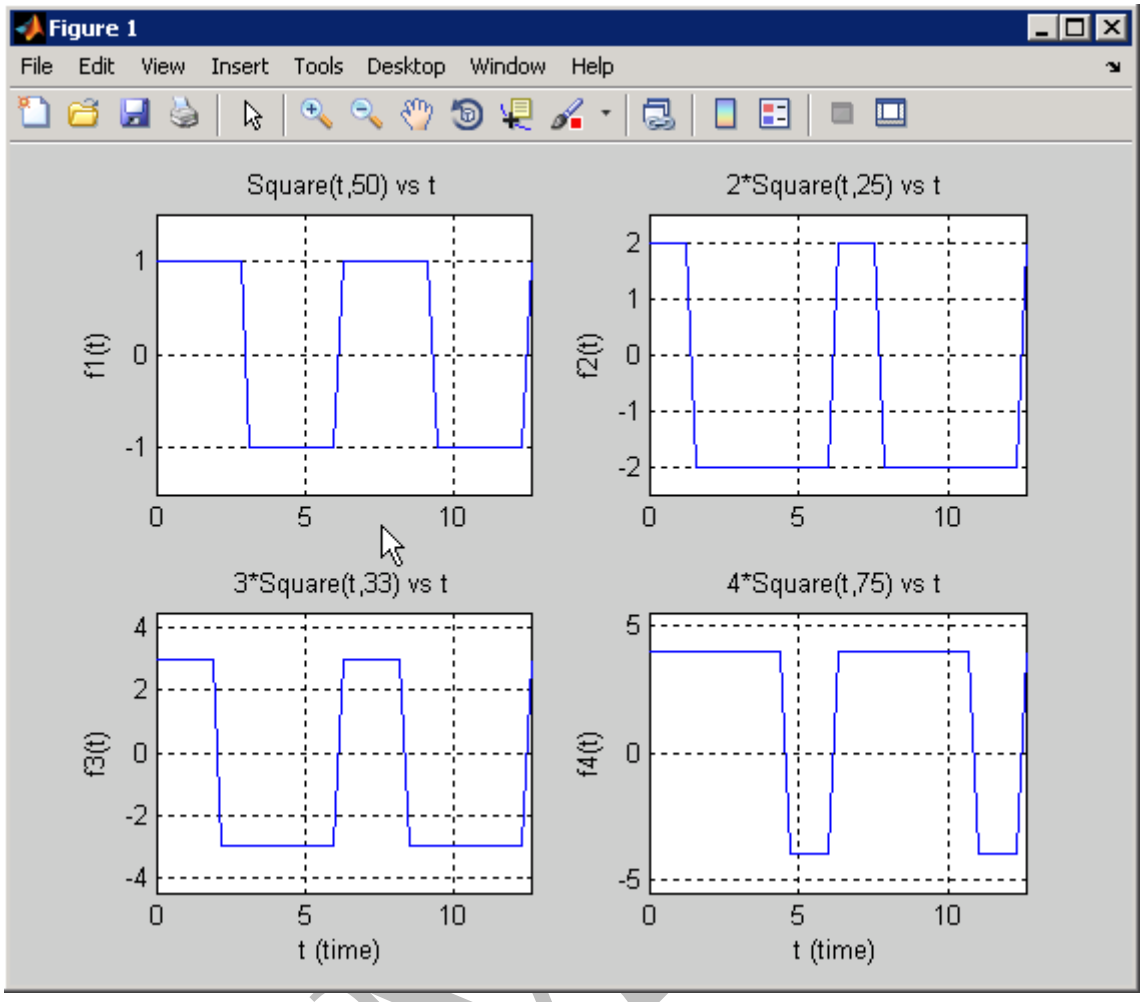
Plot discrete signal functions

We already saw how to make a Dirac (page 313) and a Heaviside (page 717) function without toolbox. Let us see now how to obtain another famous signal using the **square( )** function:



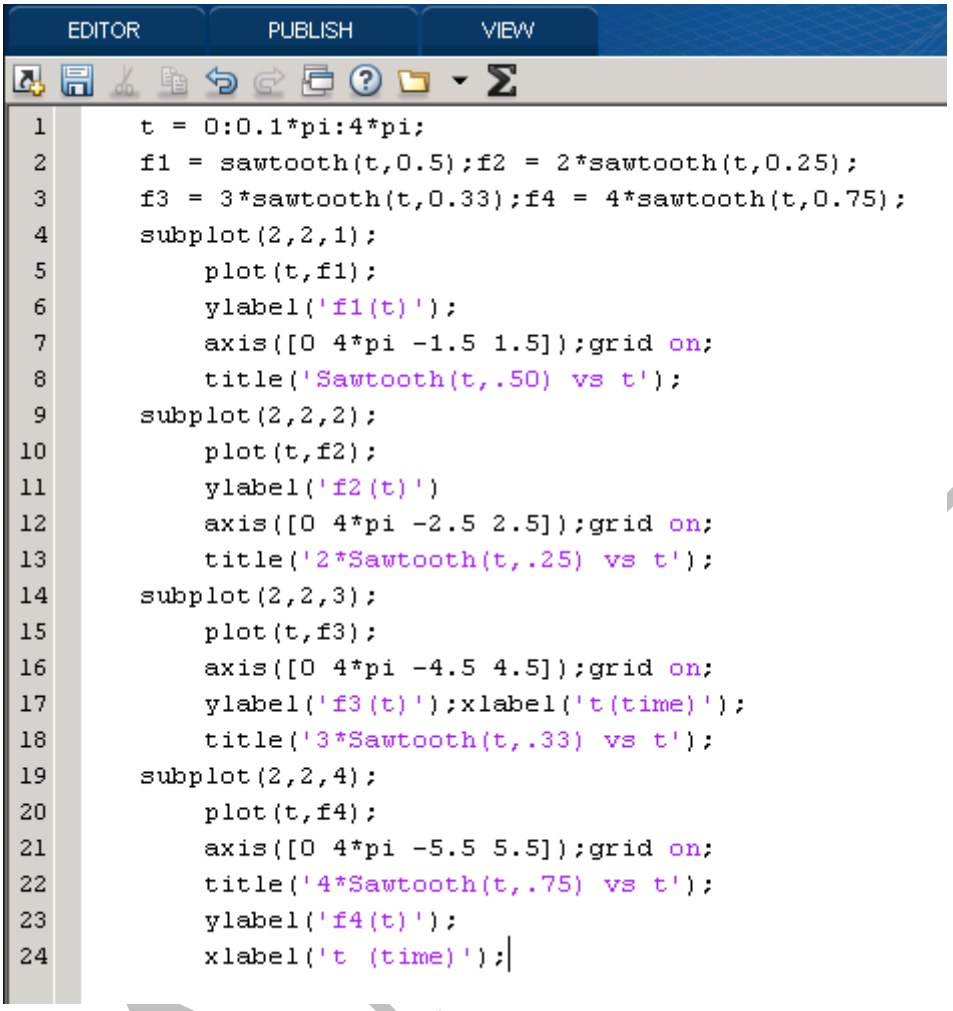

```
1   t = 0:0.1*pi:4*pi;
2   f1 =square(t,50);f2 =2*square(t,25);
3   f3 =3*square(t,33);f4 =4*square(t,75);
4   subplot(2,2,1);
5       plot(t,f1);
6       ylabel('f1(t)');
7       axis([0 4*pi -1.5 1.5]);grid on;
8       title('Square(t,50) vs t');
9   subplot(2,2,2);
10      plot(t,f2);
11      ylabel('f2(t)');
12      axis([0 4*pi -2.5 2.5]);grid on;
13      title('2*Square(t,25) vs t');
14  subplot(2,2,3);
15      plot(t,f3);
16      ylabel('f3(t)');
17      axis([0 4*pi -4.5 4.5]);grid on;
18      xlabel('t (time)')
19      title('3*Square(t,33) vs t')
20  subplot(2,2,4);
21      plot(t,f4)
22      axis([0 4*pi -5.5 5.5]);grid on;
23      title('4*Square(t,75) vs t')
24      ylabel(' f4(t) ');xlabel(' t (time) ')
```

So you get:


Figure 1
File Edit View Insert Tools Desktop Window Help
Square(t,50) vs t
2*Square(t,25) vs t
3*Square(t,33) vs t
4*Square(t,75) vs t
f1(t)
f2(t)
f3(t)
f4(t)
t (time)
t (time)

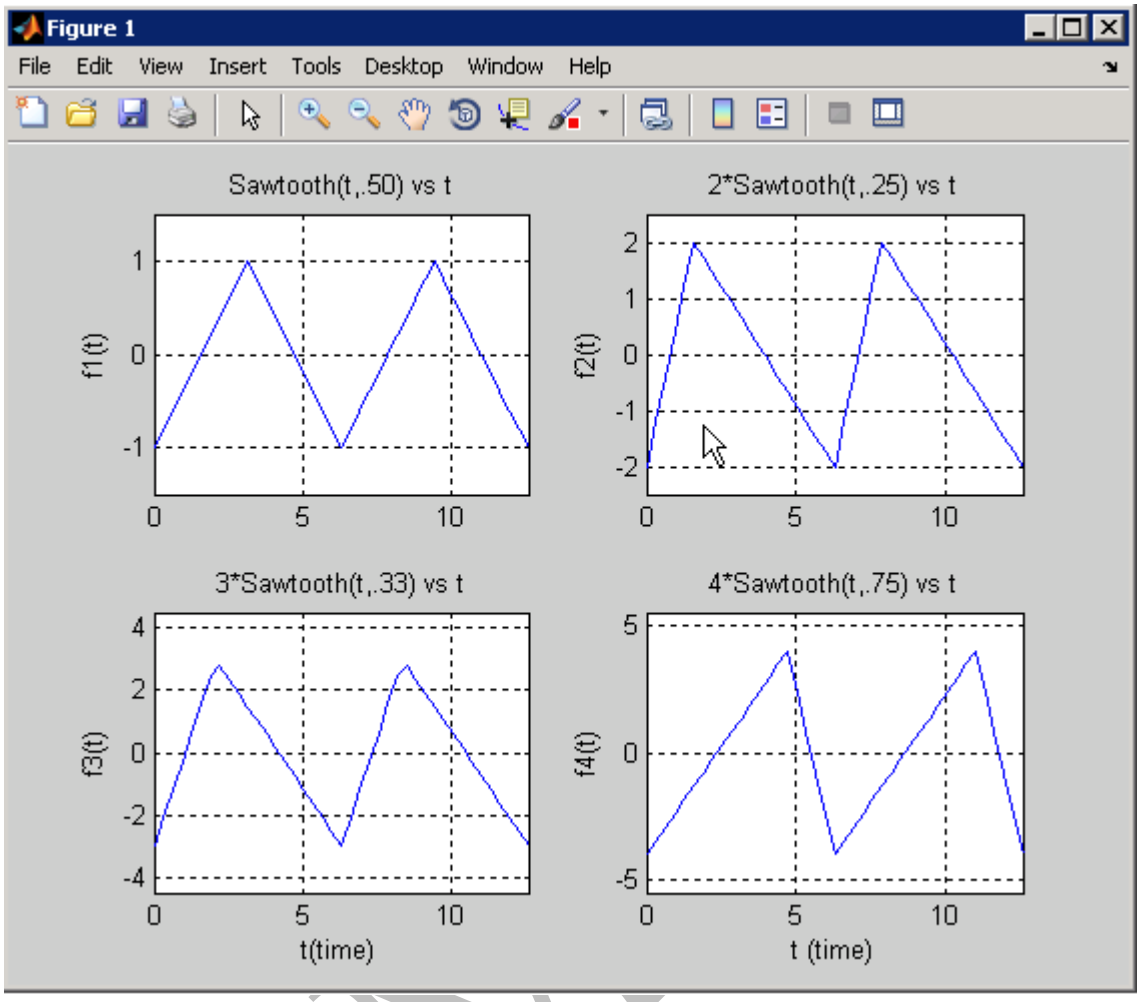
SAM

Another typical signal is the sawtooth. Here is a small example of this signal usign the **sawtooth( )** function:




```
t = 0:0.1*pi:4*pi;
f1 = sawtooth(t,0.5);f2 = 2*sawtooth(t,0.25);
f3 = 3*sawtooth(t,0.33);f4 = 4*sawtooth(t,0.75);
subplot(2,2,1);
    plot(t,f1);
    ylabel('f1(t)');
    axis([0 4*pi -1.5 1.5]);grid on;
    title('Sawtooth(t,.50) vs t');
subplot(2,2,2);
    plot(t,f2);
    ylabel('f2(t)')
    axis([0 4*pi -2.5 2.5]);grid on;
    title('2*Sawtooth(t,.25) vs t');
subplot(2,2,3);
    plot(t,f3);
    axis([0 4*pi -4.5 4.5]);grid on;
    ylabel('f3(t)');xlabel('t(time)');
    title('3*Sawtooth(t,.33) vs t');
subplot(2,2,4);
    plot(t,f4);
    axis([0 4*pi -5.5 5.5]);grid on;
    title('4*Sawtooth(t,.75) vs t');
    ylabel('f4(t)');
    xlabel('t (time)');
```

So we get:



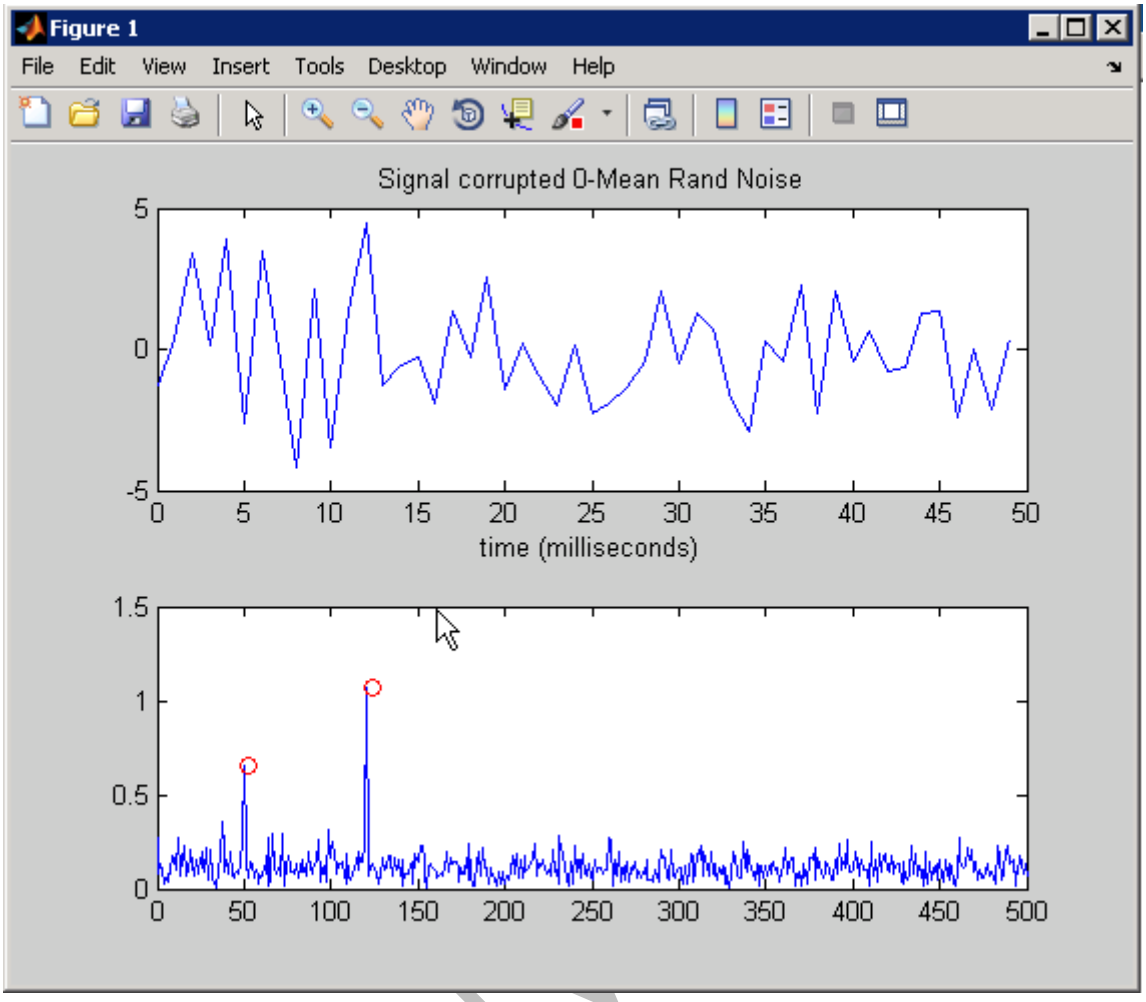
Found the peaks of signal

This a major concern for various industrial fields (finance, SPC, signal processing, etc.). For this example consider the FFT we made long ago and we want to found and plot three peaks (if they exists!) giving some constraints on the peaks using the function **findpeaks( )**:

EDITOR PUBLISH VIEW

```
1  Fs=1000;
2  T=1/Fs;
3  L=1000;
4  t=(0:L-1)*T;
5  x=0.7*sin(2*pi*50*t)+sin(2*pi*120*t);
6  y=x+2*randn(size(t));
7  subplot(2,1,1);plot(Fs*t(1:50),y(1:50));
8  title('Signal corrupted 0-Mean Rand Noise');xlabel('time (milliseconds)');
9  NFFT=2^nextpow2(L);
10 Y=fft(y,NFFT)/L;
11 f=Fs/2*linspace(0,1,NFFT/2+1);
12 subplot(2,1,2);plot(f,2*abs(Y(1:NFFT/2+1)));
13 FFT=2*abs(Y(1:NFFT/2+1));
14 [pks,locs] = findpeaks(FFT,'NPEAKS',3,'MINPEAKHEIGHT',0.5,'MINPEAKDISTANCE',10)
15 hold on;
16 plot(locs,pks,'ro');
```

So we get:



And in the *Command Window* we have the values and positions of the peaks.

Add echo to a sound (1D digital filter)

Echoes are delayed signals, and as such are generated using delay units. For example, the combination of the direct sound represented by discrete signal $y[n]$ and a single echo appearing D samples later (which is related to delay in seconds) with an amplitude volume factor of α can be generated by the equation of the form (called a difference equation):

$$x[n] = y[n] + \alpha y[n - D]$$

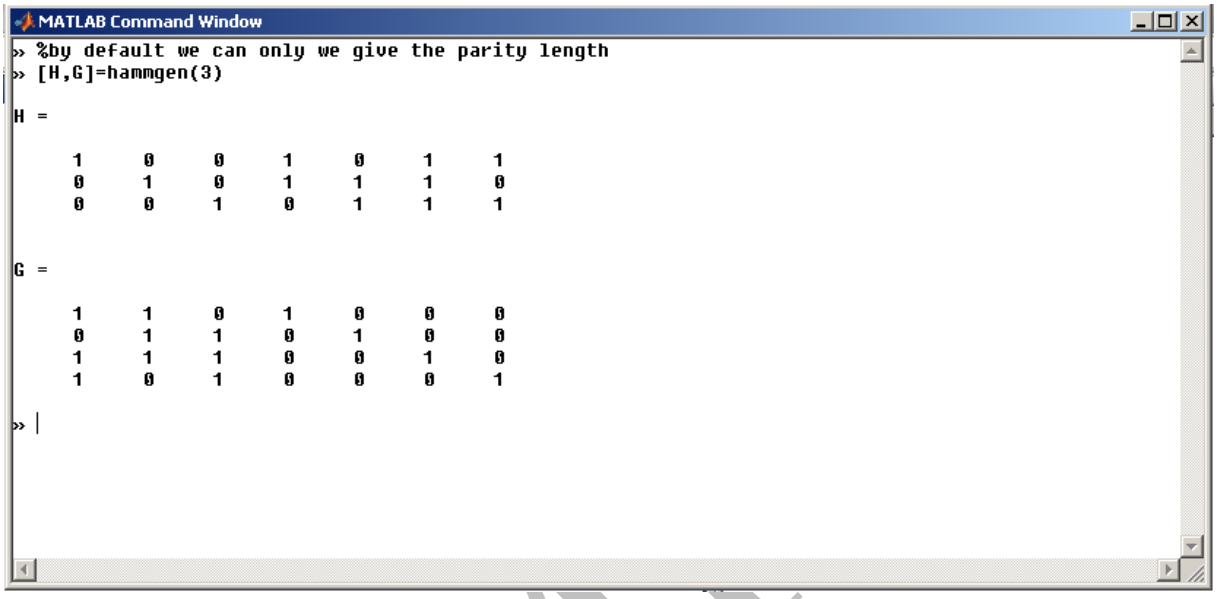
EDITOR PUBLISH VIEW

```
1    load handel; %the signal "Alleluia song" is in y and sampling freq in Fs
2    sound(y,Fs); pause(10); %Play the original sound
3    alpha = 0.9; D = 4196; %Echo parameters
4    b = [1,zeros(1,D),alpha]; %Filter parameters
5    x = filter(b,1,y); %Generate sound plus its echo
6    sound(x,Fs); %Play sound with echo
```

To remove the echo we use after if necessary we invert the parameters of the **filter( )** command: w = filter(1,b,x);sound(w,Fs)

Hamming Error Correcting Code

The purpose here is to use MATLAB to see if we can build quickly the parity check and generator matrices of the Hamming code introduced in the theoretical course:

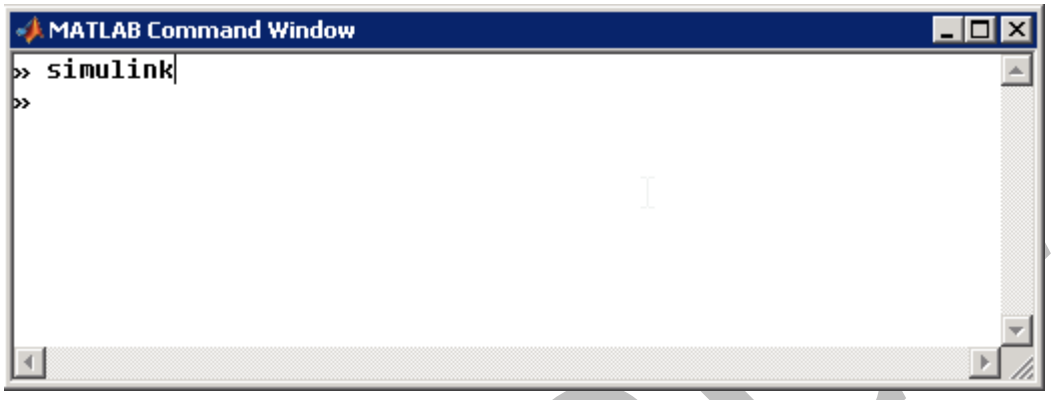


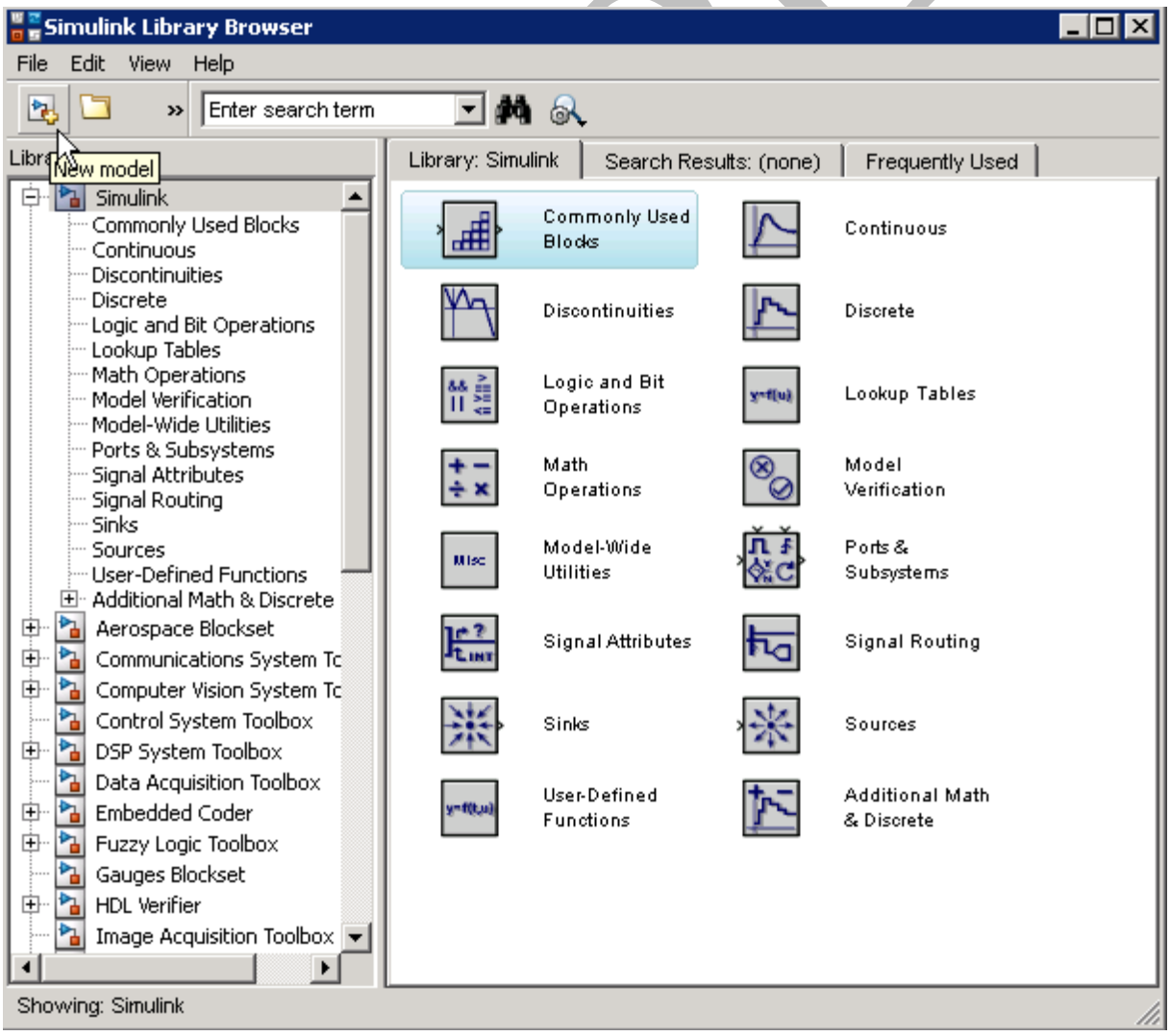
This is therefore the Hamming(7,4) code. It is a linear error-correcting code that encodes four bits of data into seven bits by adding three parity bits. It is a member of a larger family of Hamming codes, but the term Hamming code often refers to this specific code that Richard W. Hamming introduced in 1950. At the time, Hamming worked at Bell Telephone Laboratories and was frustrated with the error-prone punched card reader, which is why he started working on error-correcting codes

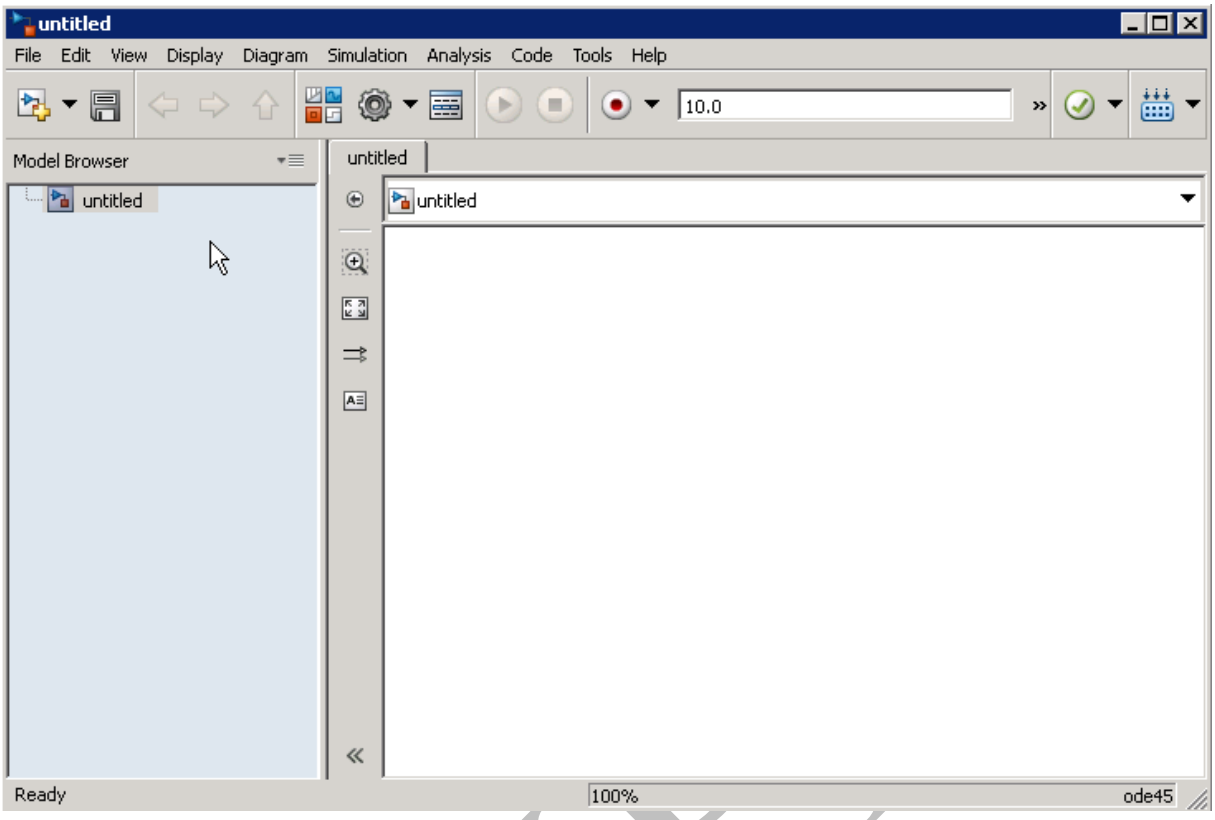
SAMPLE

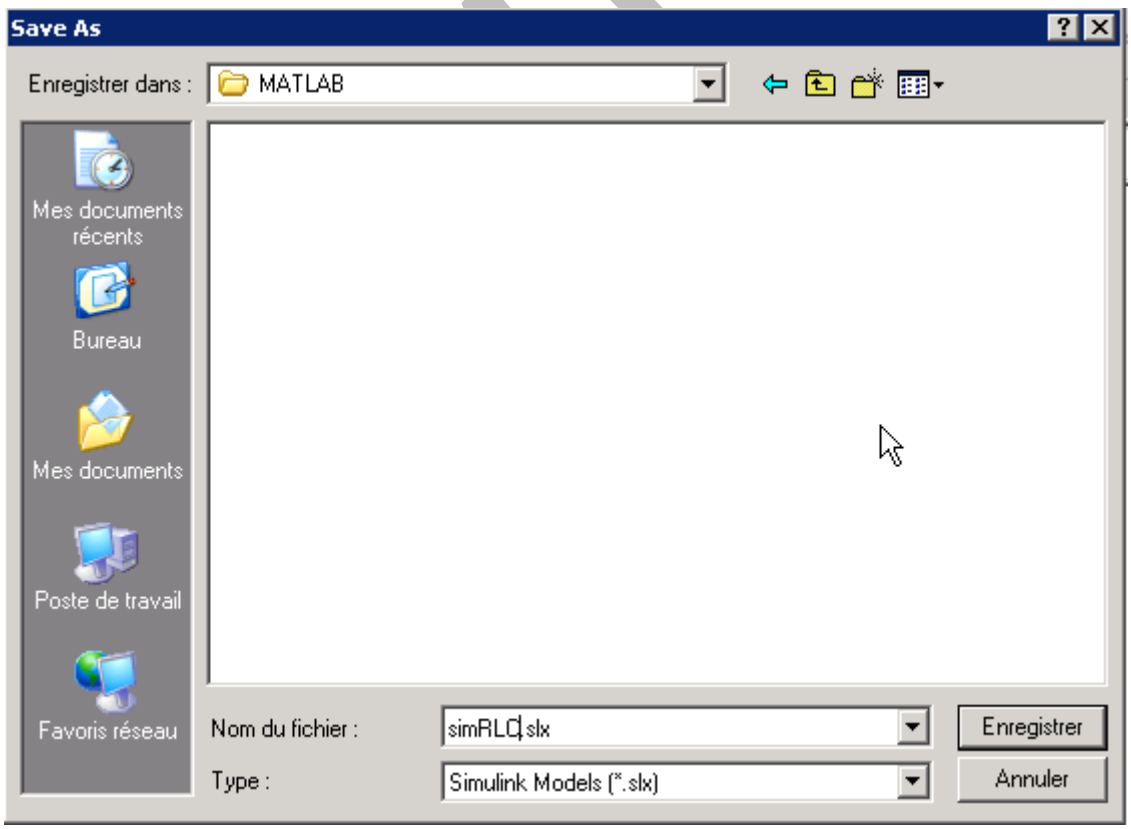
Circuit Analysis (Simulink Toolbox)

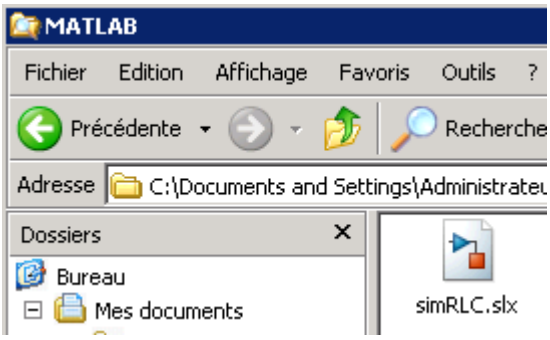
Simulink, developed by The MathWorks, is a commercial tool for modeling, simulating and analyzing dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for simulation and design.



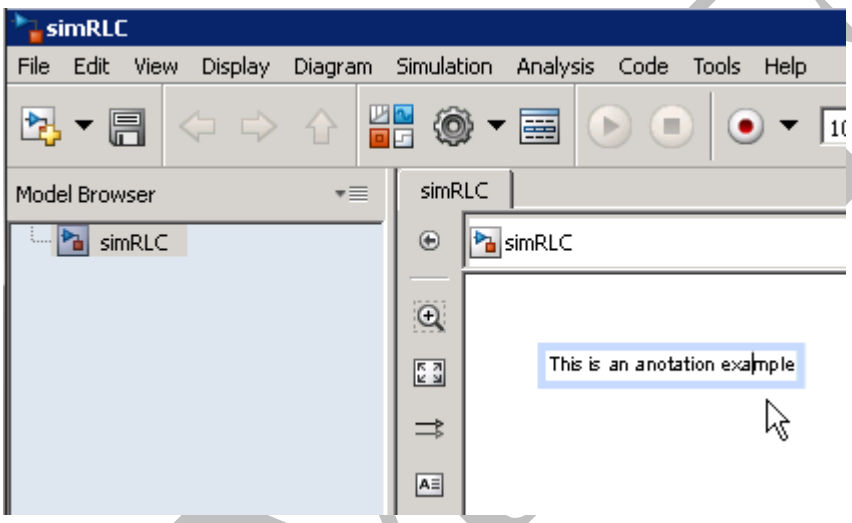



untitled
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Model Browser
untitled
untitled
untitled
Ready
100%
ode45


Save As
Enregistrer dans :
MATLAB
Mes documents récents
Bureau
Mes documents
Poste de travail
Favoris réseau
Nom du fichier :
simRLC.slx
Type :
Simulink Models (*.slx)
Enregistrer
Annuler

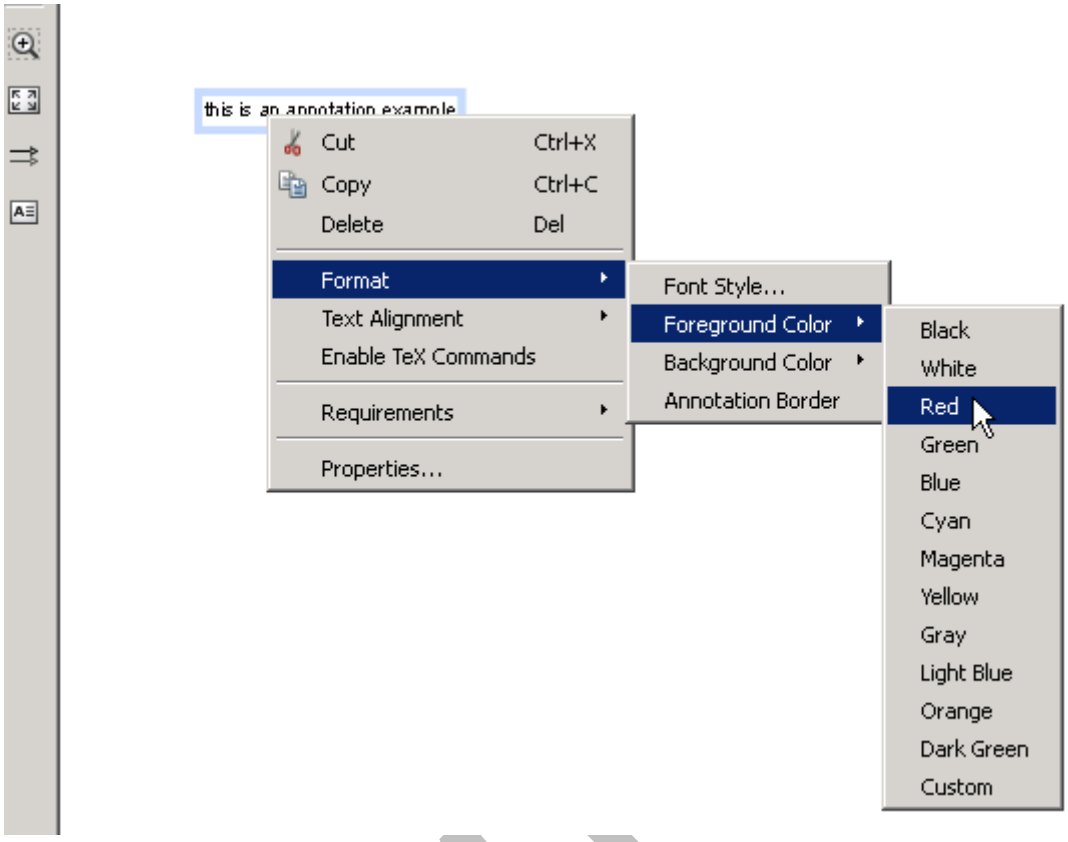


The firs thing you have to know is how to annotate. For this purpose just double click anyhwere in the background:

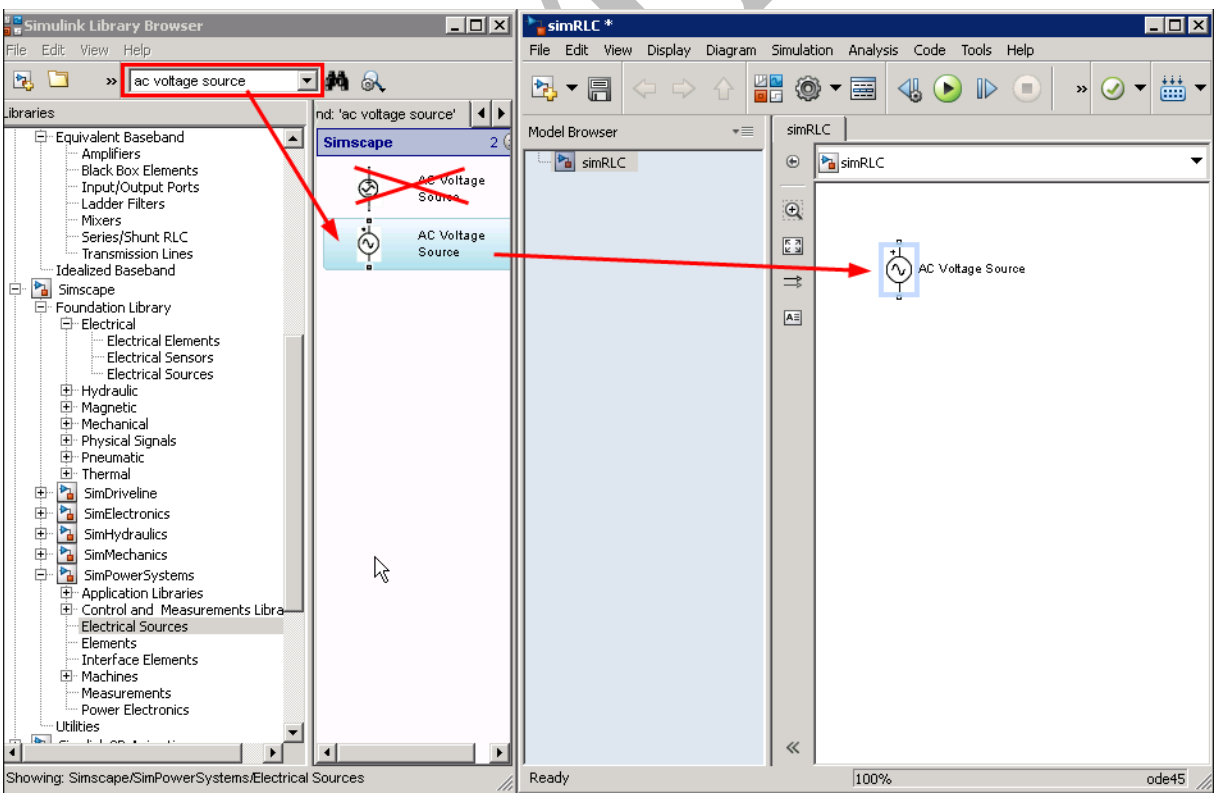


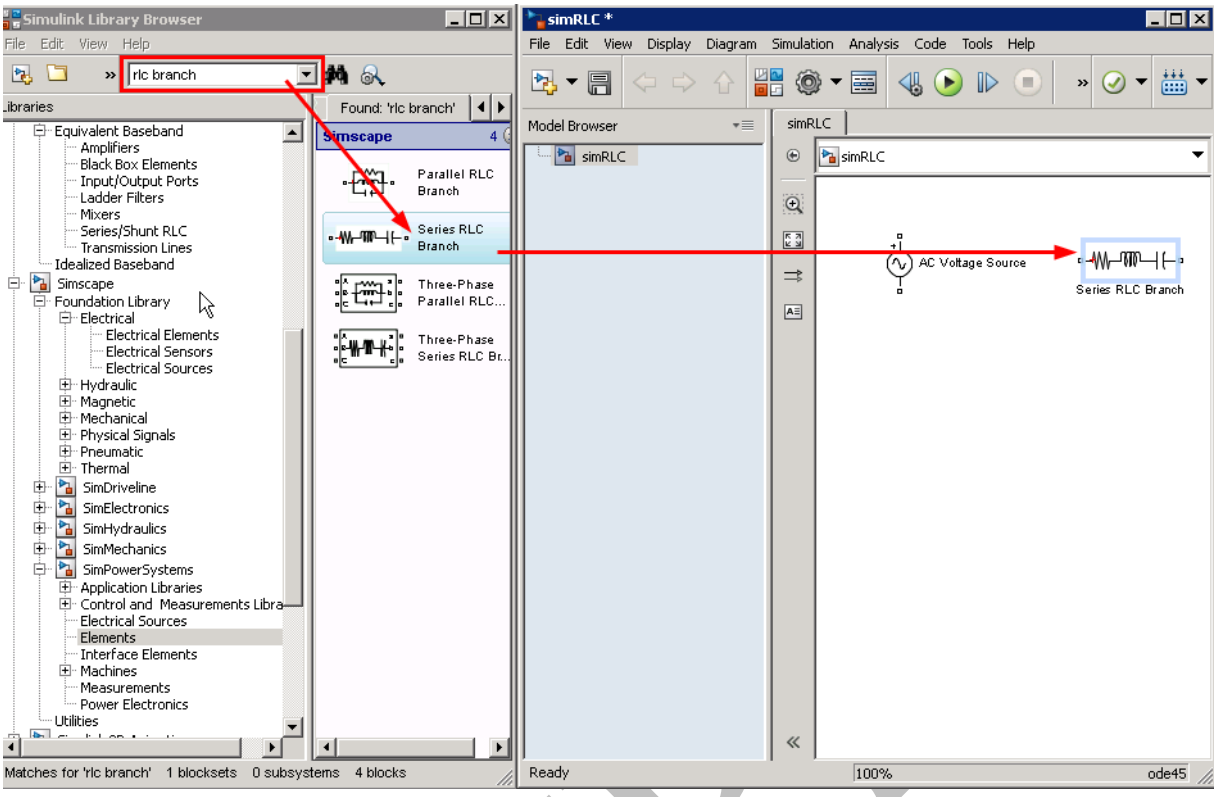
and to change the colors of the text (or of anything else):

SAMPLE



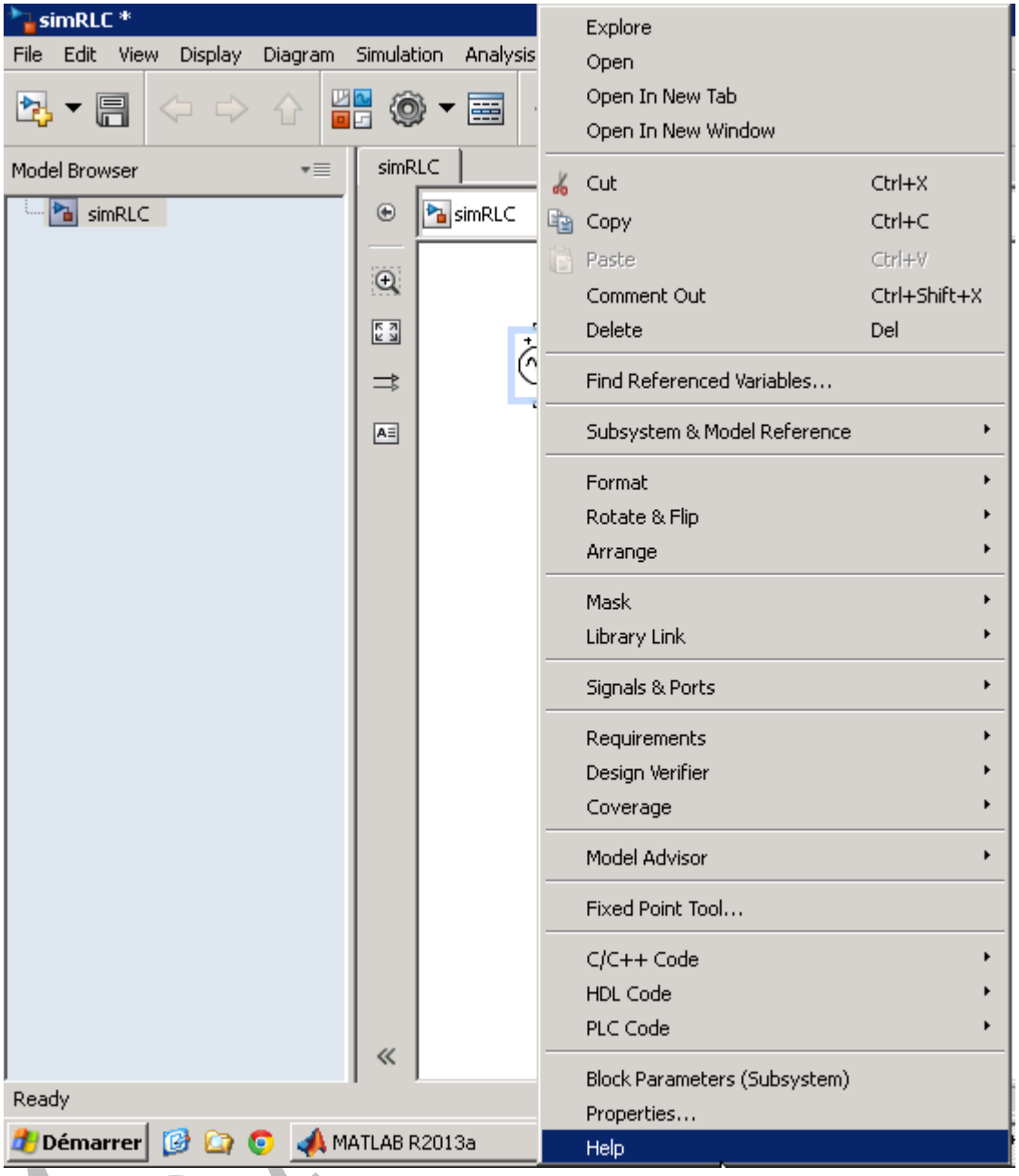
Don't the first one or you won't be able to create wires later!

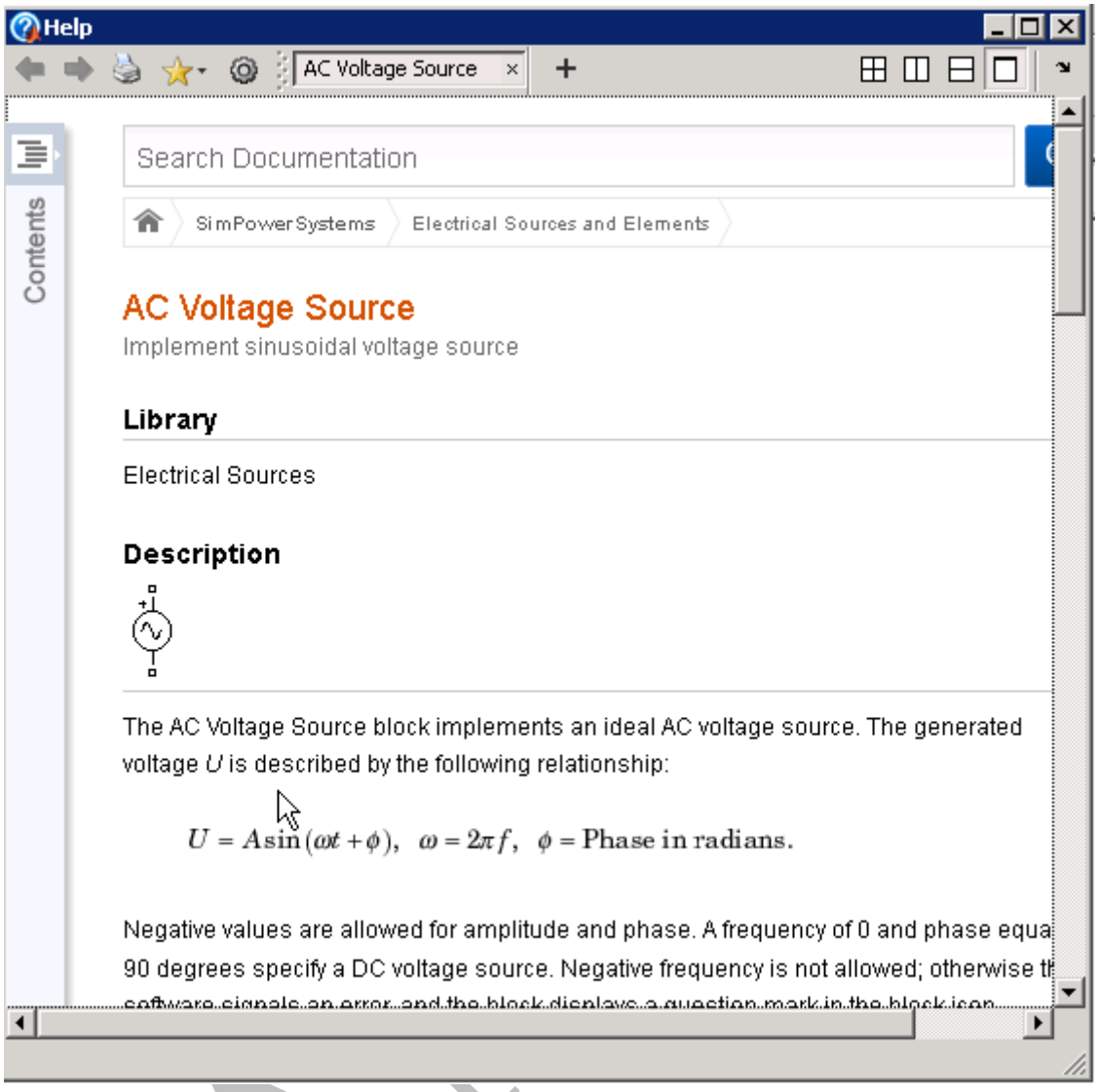




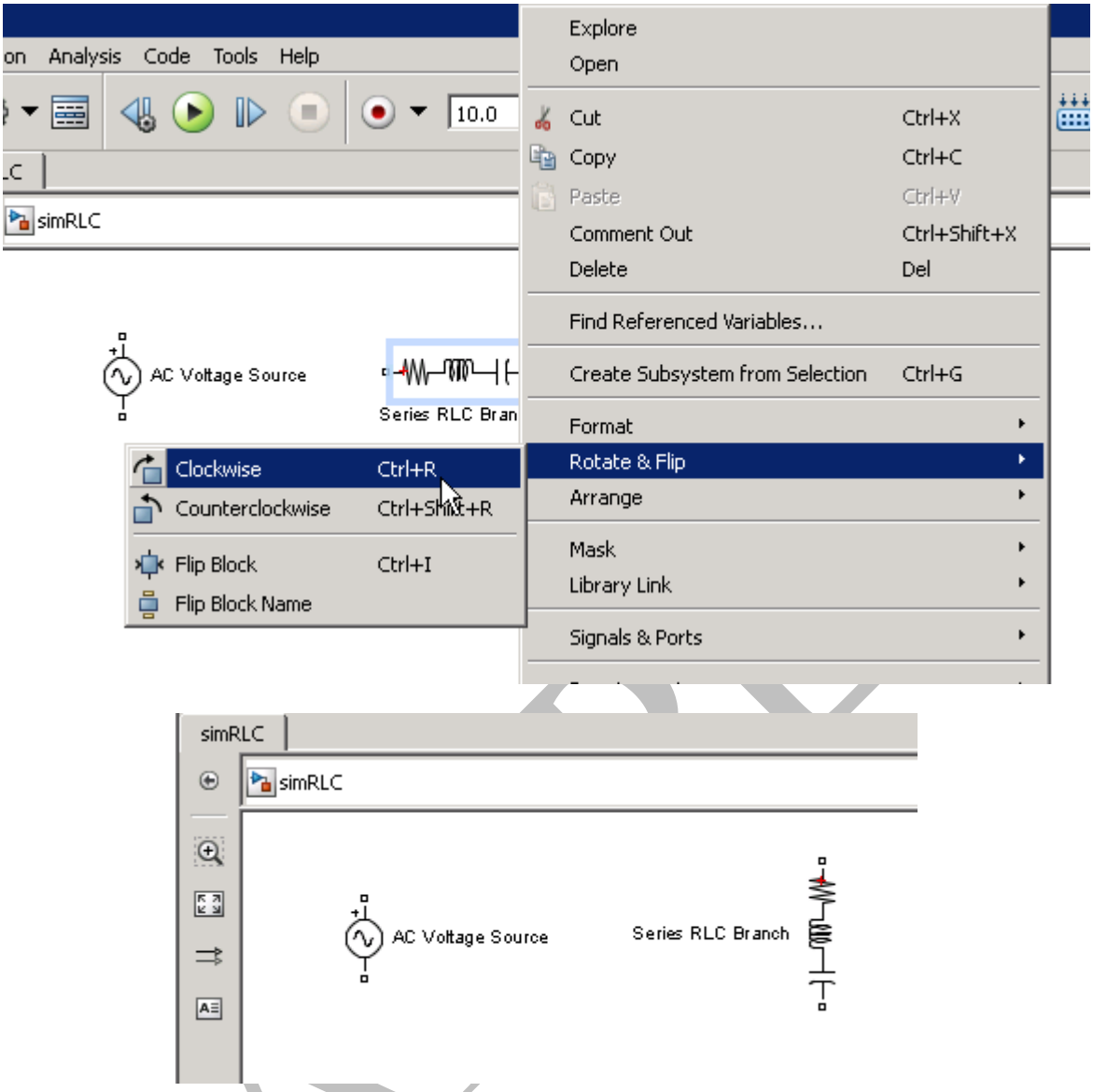
All standard blocks in Simulink have detailed Help. Click the Help button in the Block Parameter window for the specific block in order to get detailed help for that block.

SAMPLE


simRLC *
File Edit View Display Diagram Simulation Analysis
Model Browser
simRLC
simRLC
simRLC
Explore
Open
Open In New Tab
Open In New Window
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Comment Out Ctrl+Shift+X
Delete Del
Find Referenced Variables...
Subsystem & Model Reference
Format
Rotate & Flip
Arrange
Mask
Library Link
Signals & Ports
Requirements
Design Verifier
Coverage
Model Advisor
Fixed Point Tool...
C/C++ Code
HDL Code
PLC Code
Block Parameters (Subsystem)
Properties...
Help
Ready
Démarrer
MATLAB R2013a


Help
AC Voltage Source
Search Documentation
Contents
SimPowerSystems
Electrical Sources and Elements
AC Voltage Source
Implement sinusoidal voltage source
Library
Electrical Sources
Description
The AC Voltage Source block implements an ideal AC voltage source. The generated voltage U is described by the following relationship:
$U = A\sin(\omega t + \phi), \quad \omega = 2\pi f, \quad \phi = \text{Phase in radians.}$
Negative values are allowed for amplitude and phase. A frequency of 0 and phase equa
90 degrees specify a DC voltage source. Negative frequency is not allowed; otherwise t
software signals an error and the block displays a question mark in the block icon.

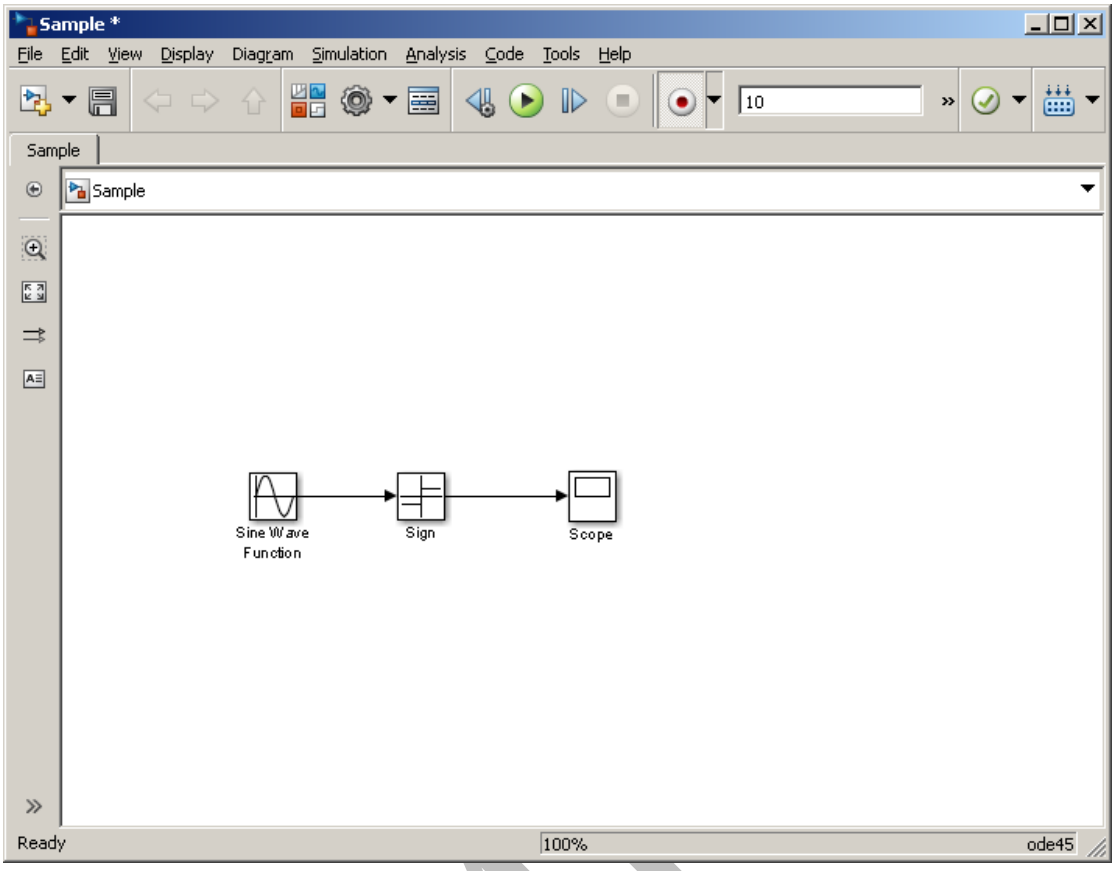
SA



If you want you can create connectors without any problems just by dragging the mouse between connector points.

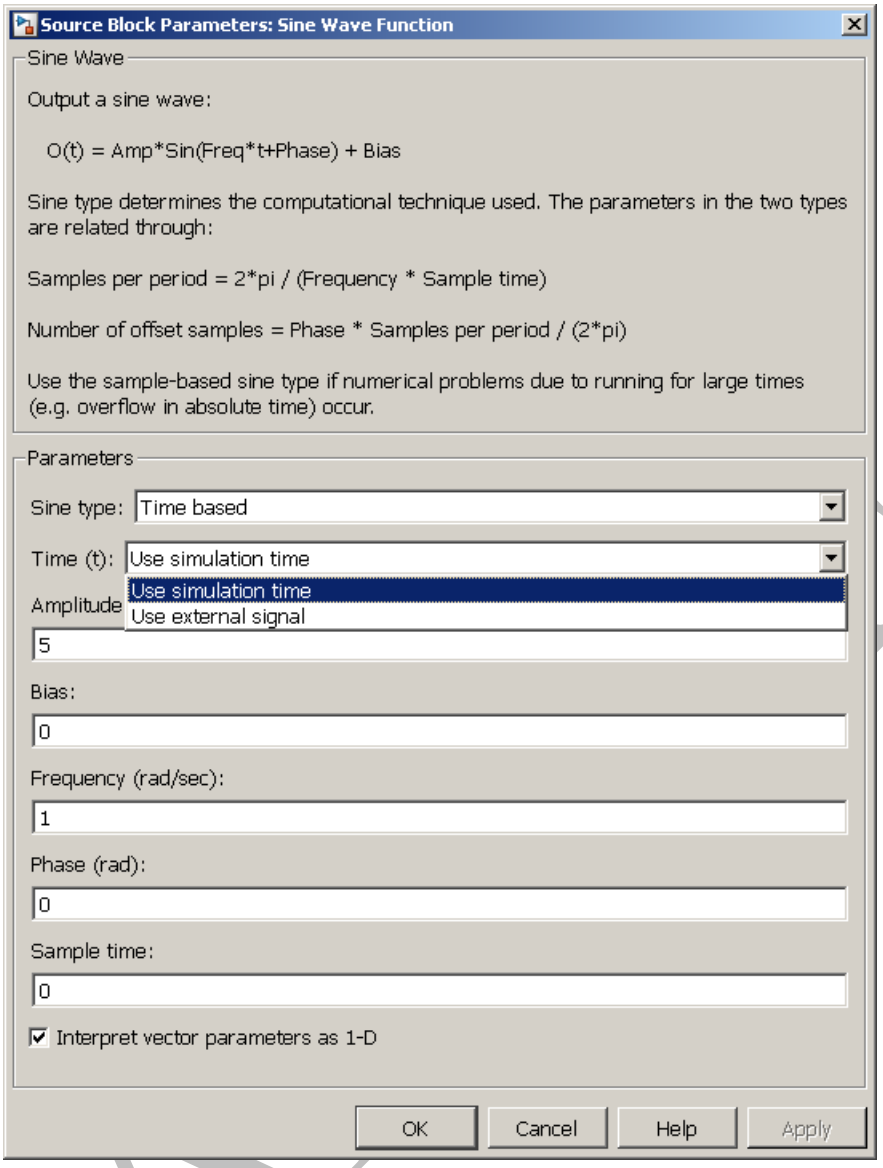
Create a simple signal filter

Consider we want to filter a sinus signal to get only the sign of a signal (using the signum function). To do this we only have to create the following circuit (I think we can not find an easier practical example ☺):

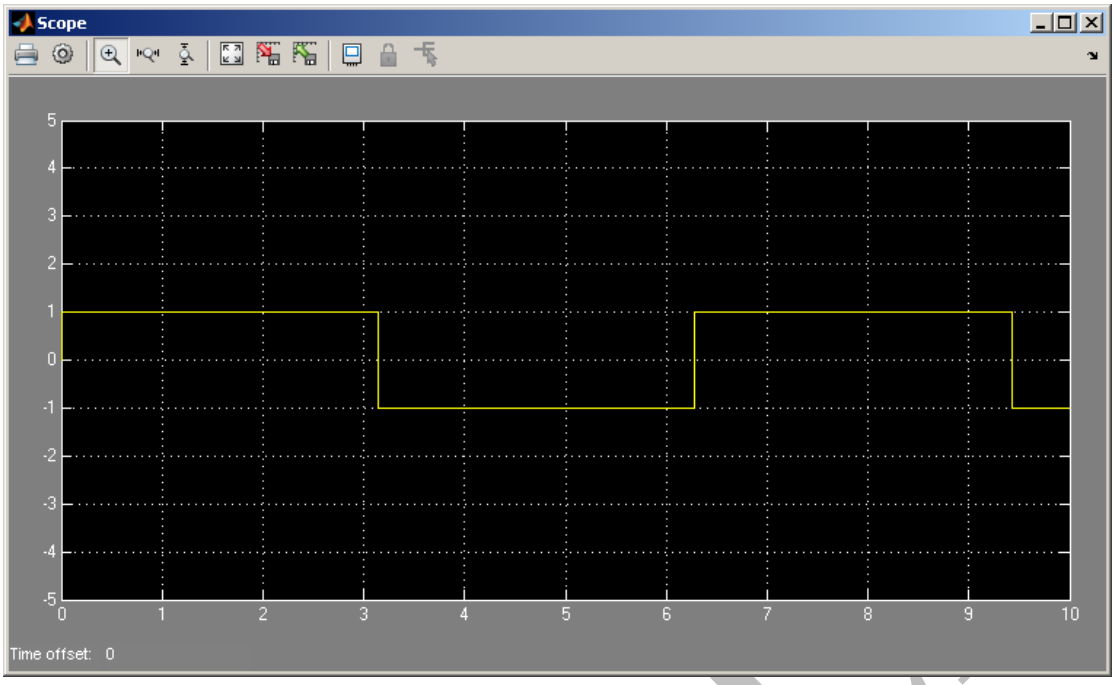


When you double click on the component **Sine Wave Function** take:

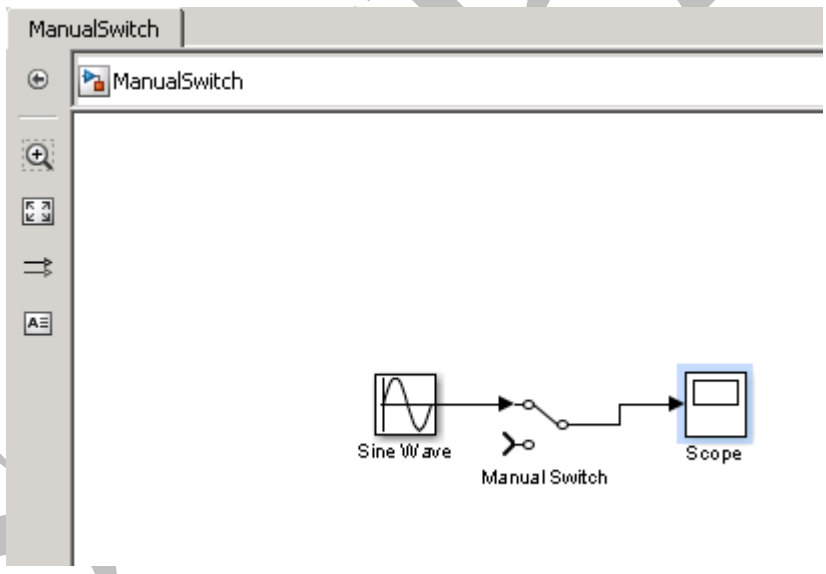
SAMPLE



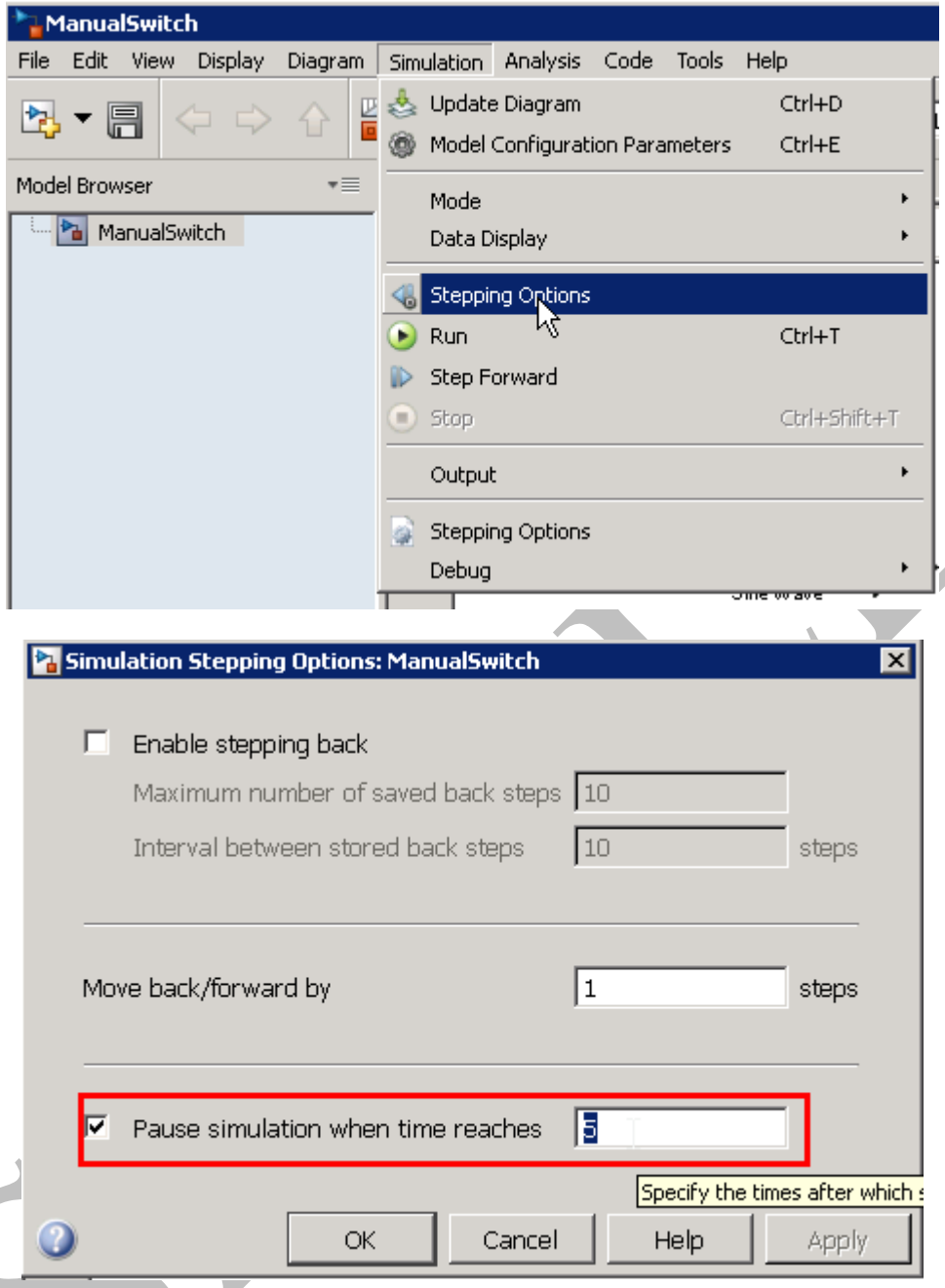
You run the simulation to get after a double click on the scope:

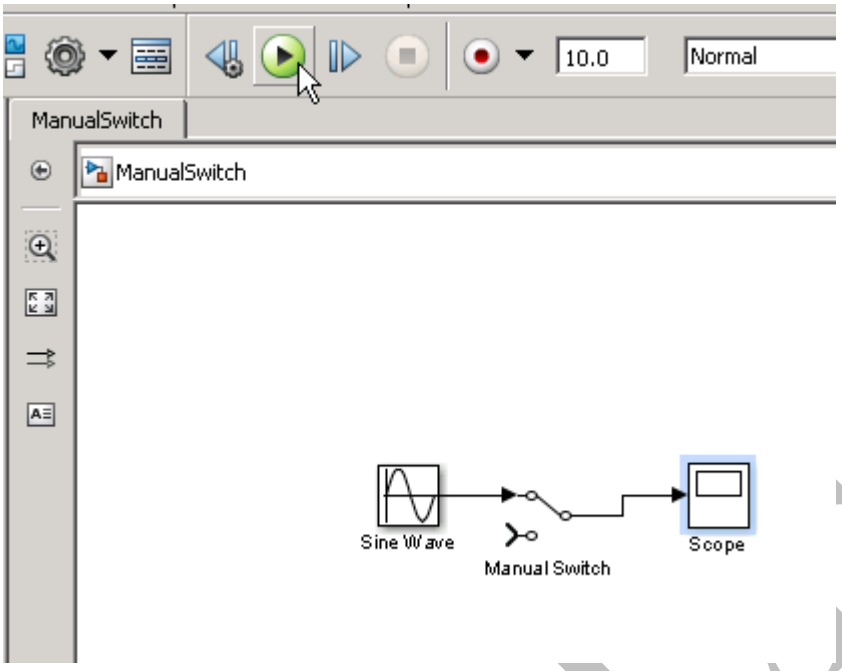


Create a simple manual switch on/switch off signal

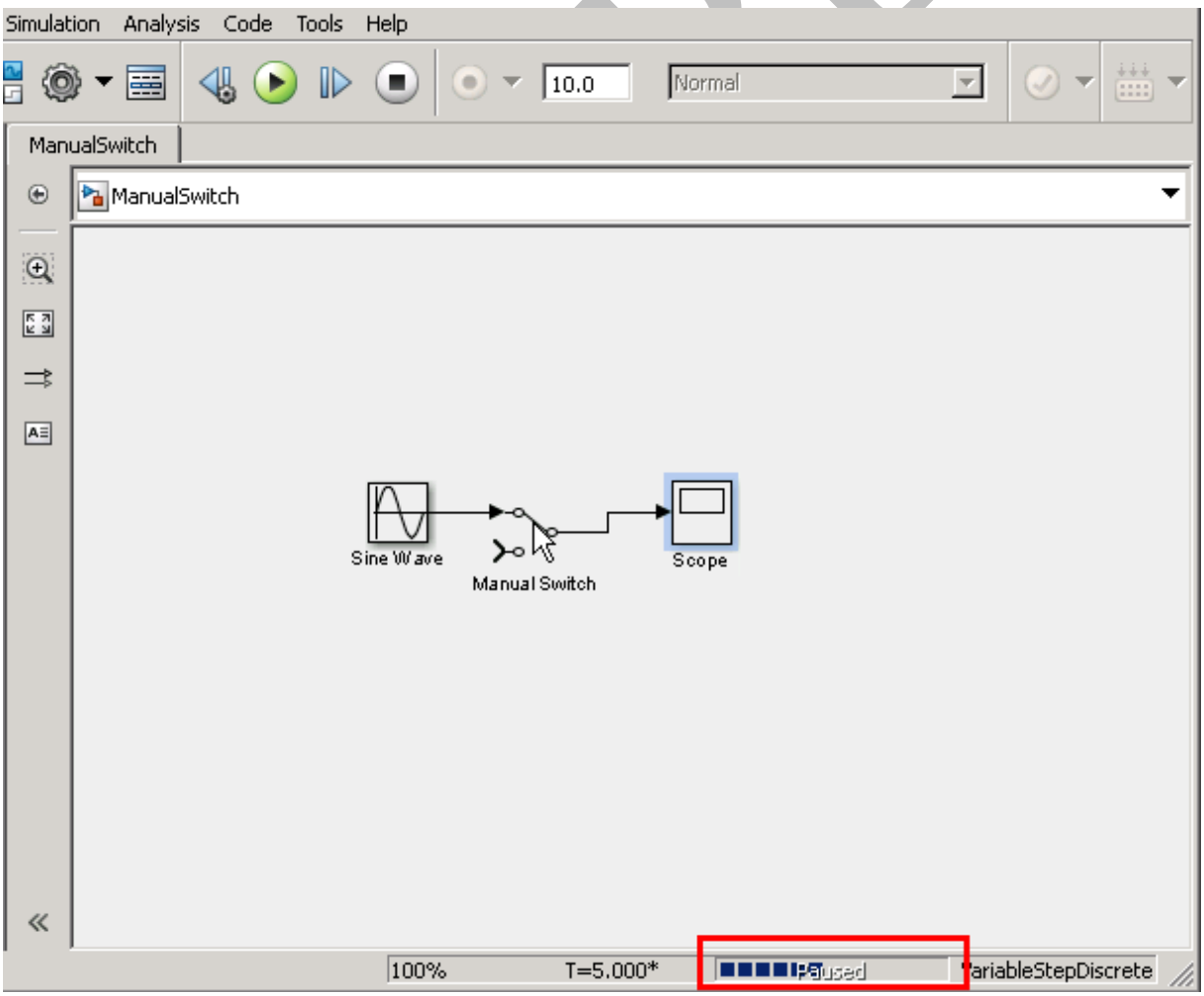


Before you run the simulation, decide when you want to change the switch status:

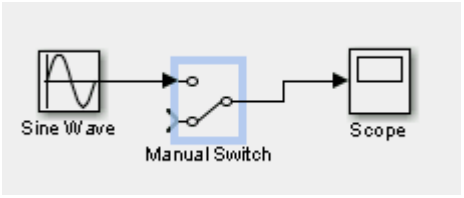

ManualSwitch
File Edit View Display Diagram Simulation Analysis Code Tools Help
Update Diagram Ctrl+D
Model Configuration Parameters Ctrl+E
Model Browser
ManualSwitch
Mode
Data Display
Stepping Options
Run Ctrl+T
Step Forward
Stop Ctrl+Shift+T
Output
Stepping Options
Debug
Simulation Stepping Options: ManualSwitch
Enable stepping back
Maximum number of saved back steps 10
Interval between stored back steps 10 steps
Move back/forward by 1 steps
Pause simulation when time reaches 5
Specify the times after which
OK Cancel Help Apply



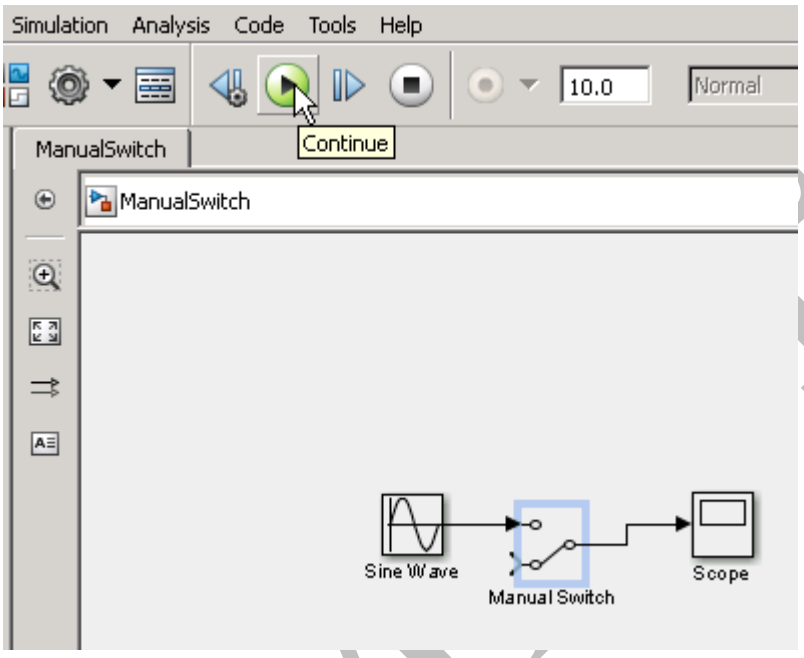
MATLAB will pause when it reaches the 5 seconds:



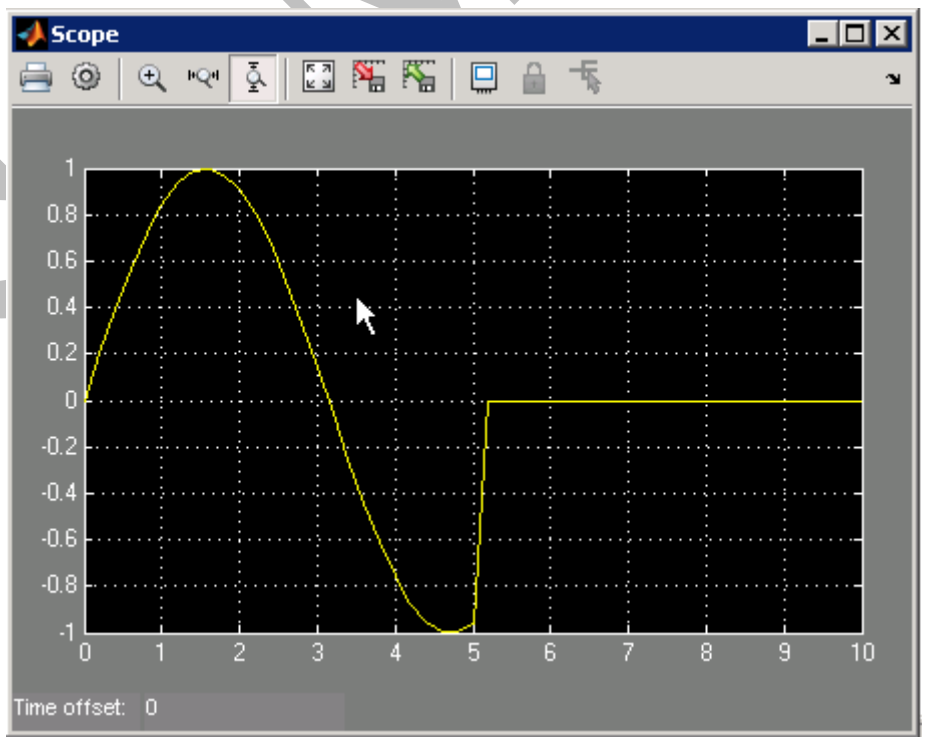
Toggle the switch by clicking on it:



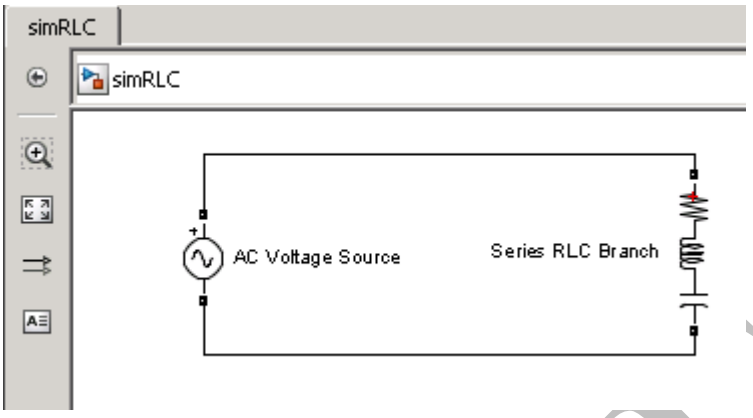
Click again on the play simulation button:



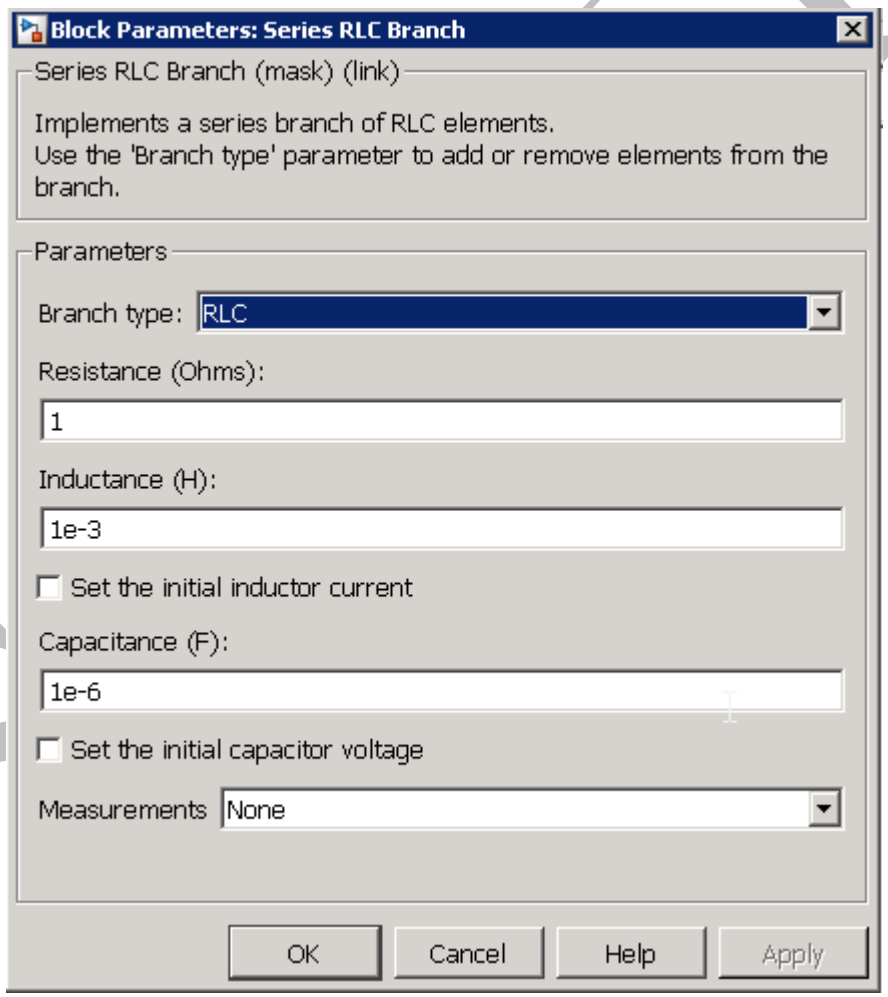
And double click on the Scope (**even if there is an error message!**) to get:

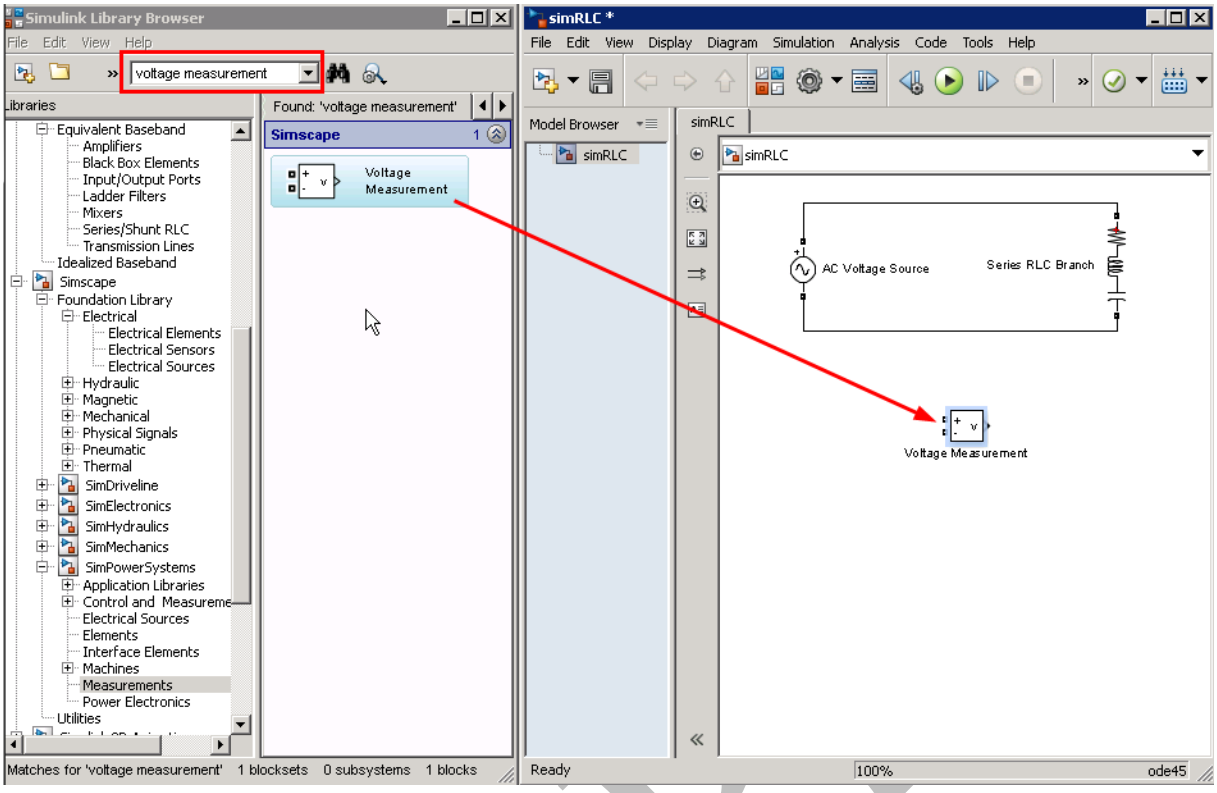


Create an RLC branch circuit

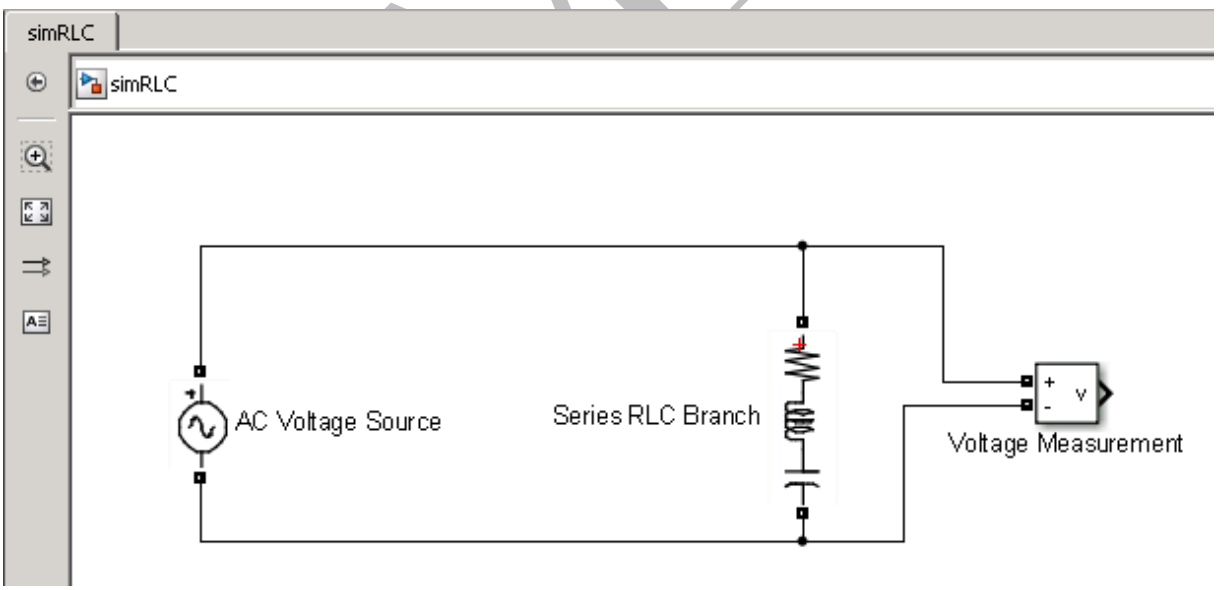


If you double click on the RLC element you get:

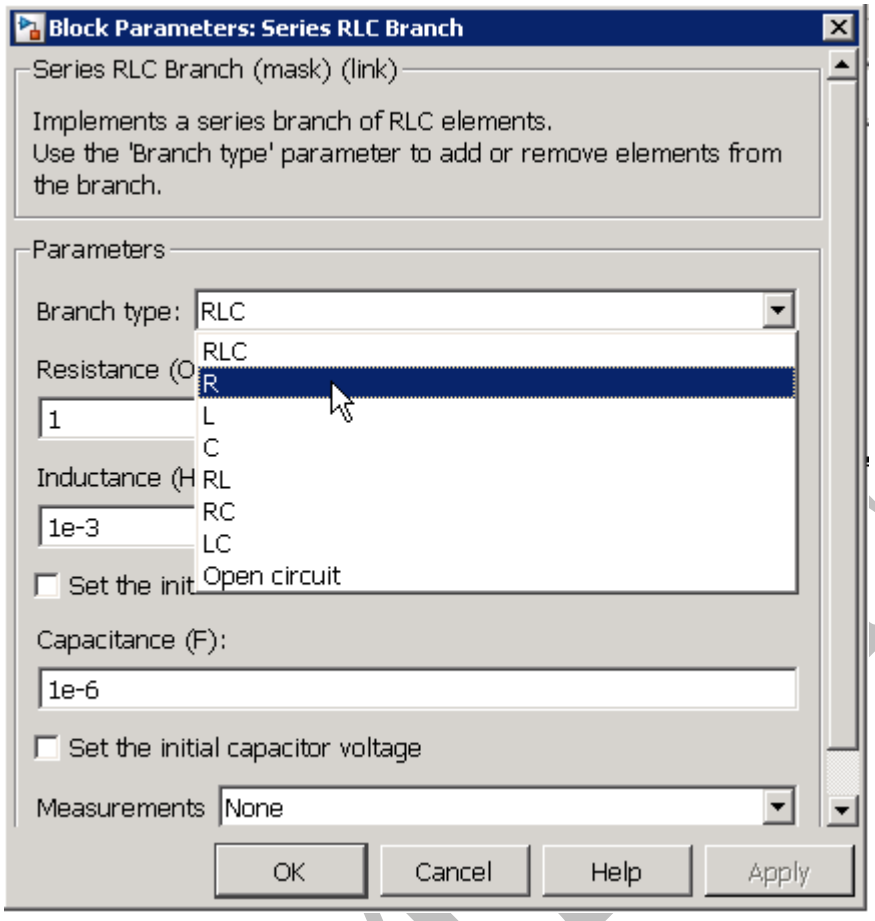


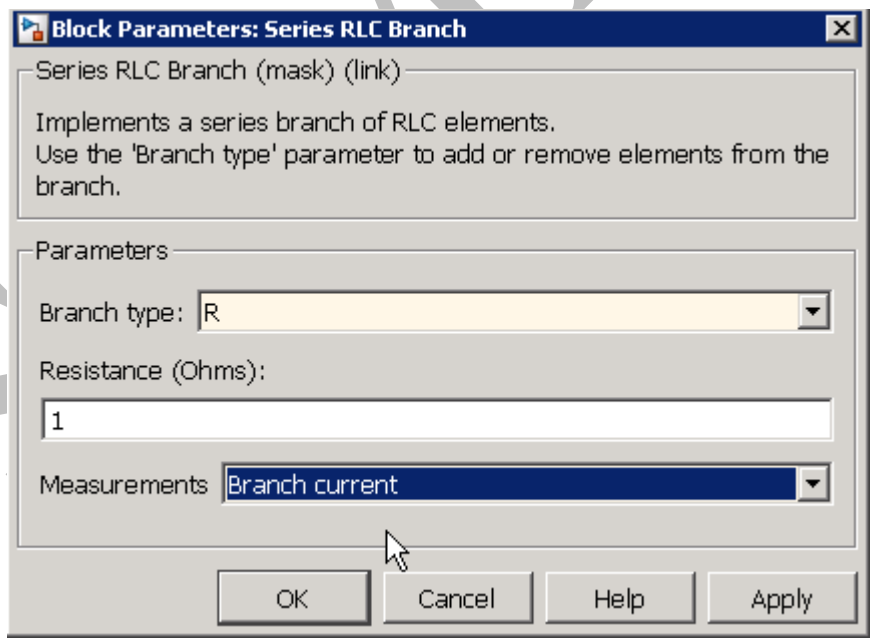


After link the output of the Voltage Measurement and connect them on the existing wire parts (you can't connect the wires between the R only or the L only or the C only with an RLC Branch):



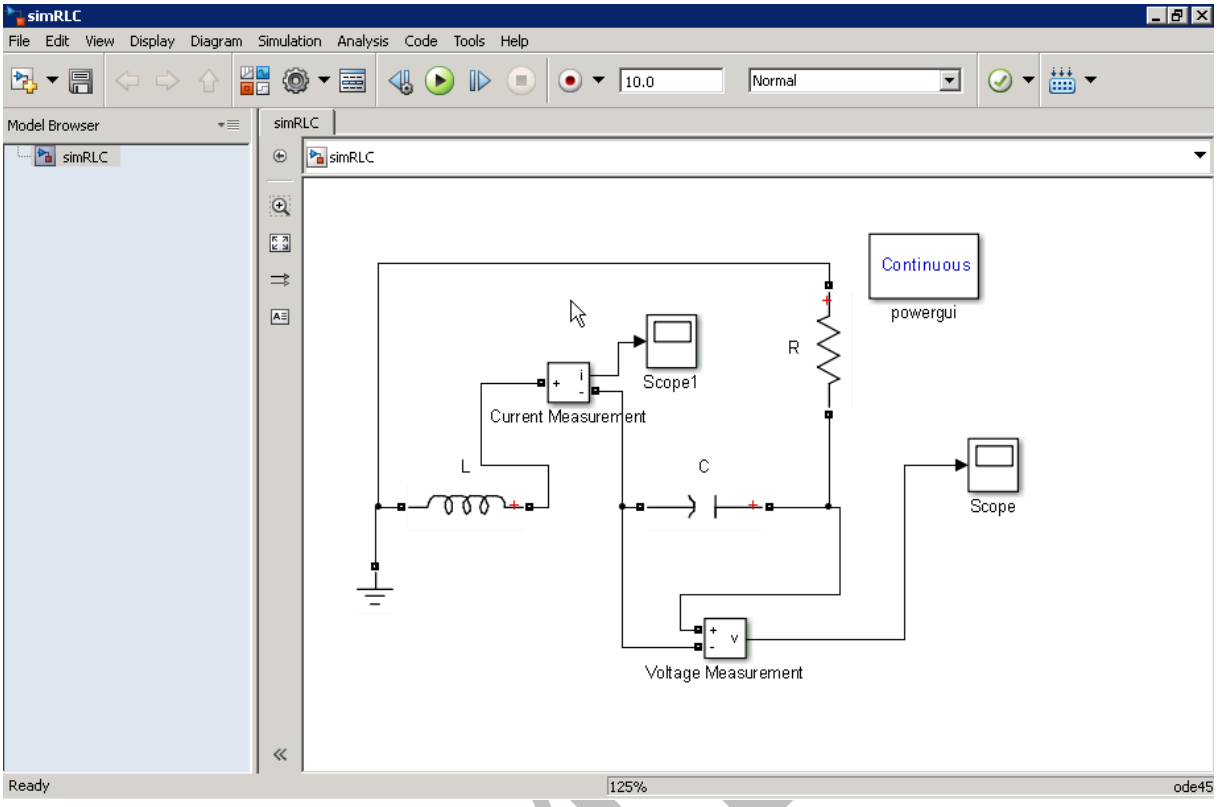
If you wan't to connect only between one of the items:



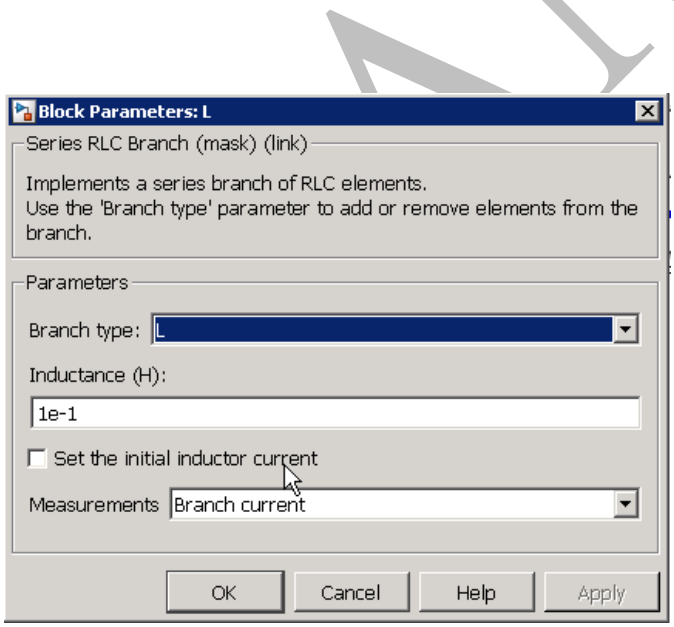


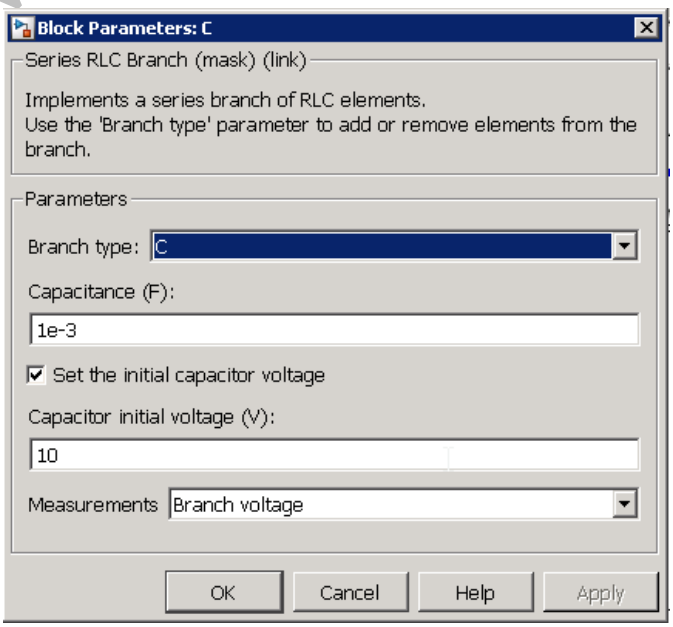
Now if we want we can do the same by using separate components and because voltage between the branch is almost useless we add a parallel voltage measurement one the capacitance and serial intensity measurement to compare results with the theory study in the theoretical course.

Following same procedure as before we build the following diagram with the idea that when we start the simulation, the start corresponds to change a manual switch status on ON:



Now we run with the standard Simulink settings three scenarios of the serial RLC by choosing for each scenario constant arbitrary values for the inductance and capacity with the following parameters:





Then in this case we have:

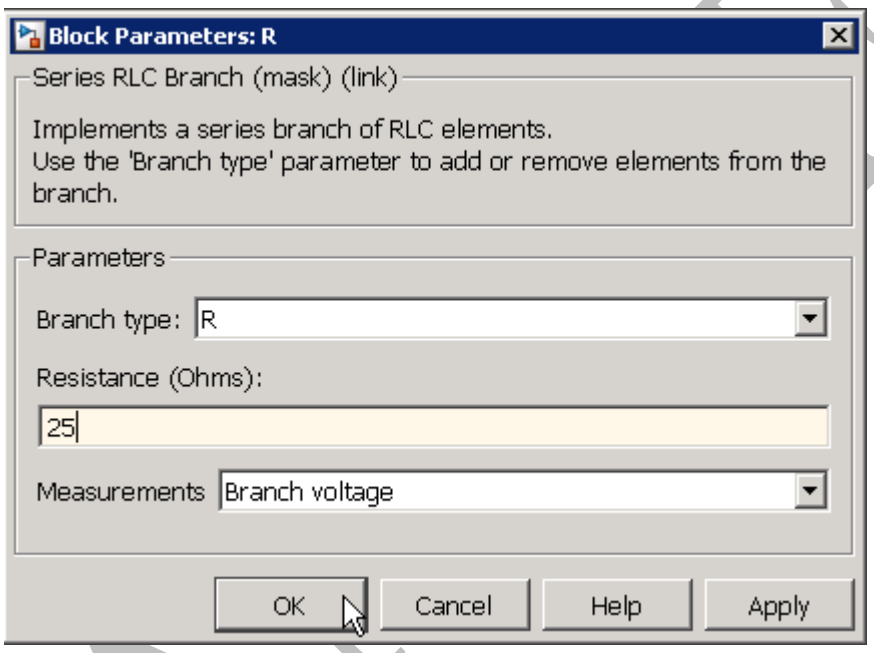
$$R_c = 2\sqrt{\frac{L}{C}} = 20$$

1. Over-damped circuit

For refresh this is the case where:

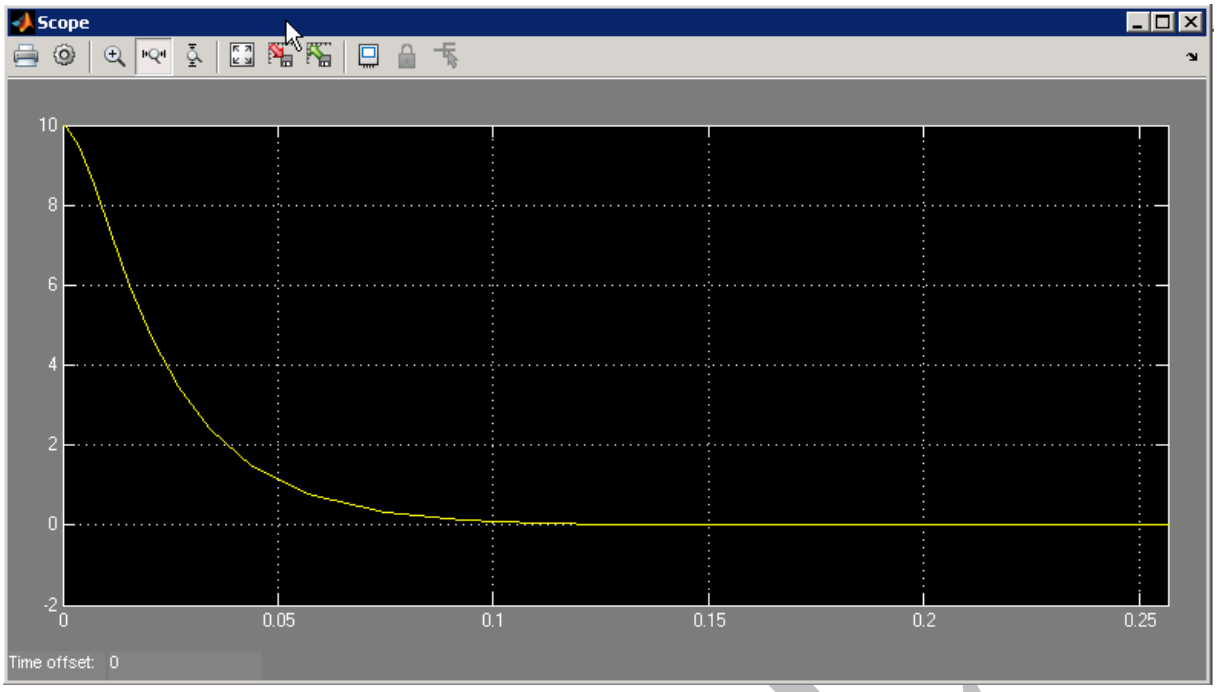
$$R > 2\sqrt{\frac{L}{C}}$$

So:

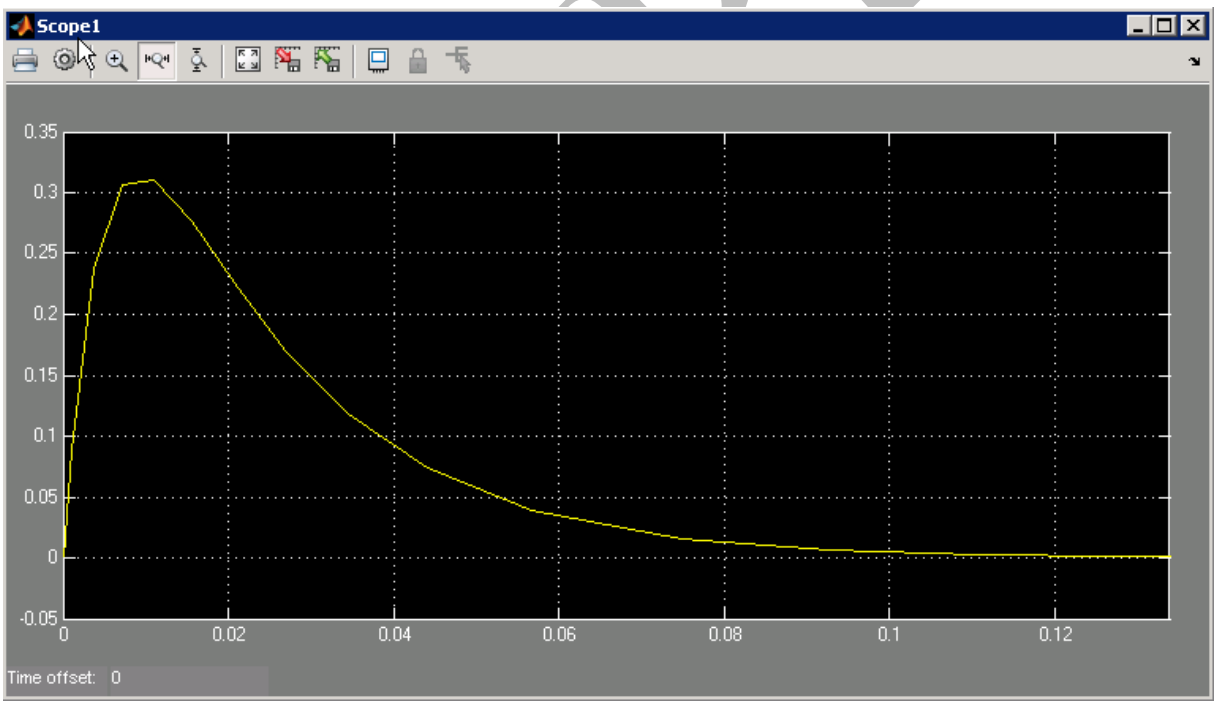


When we run the simulation and we double click on the voltage measurement we get:

SAMPLE



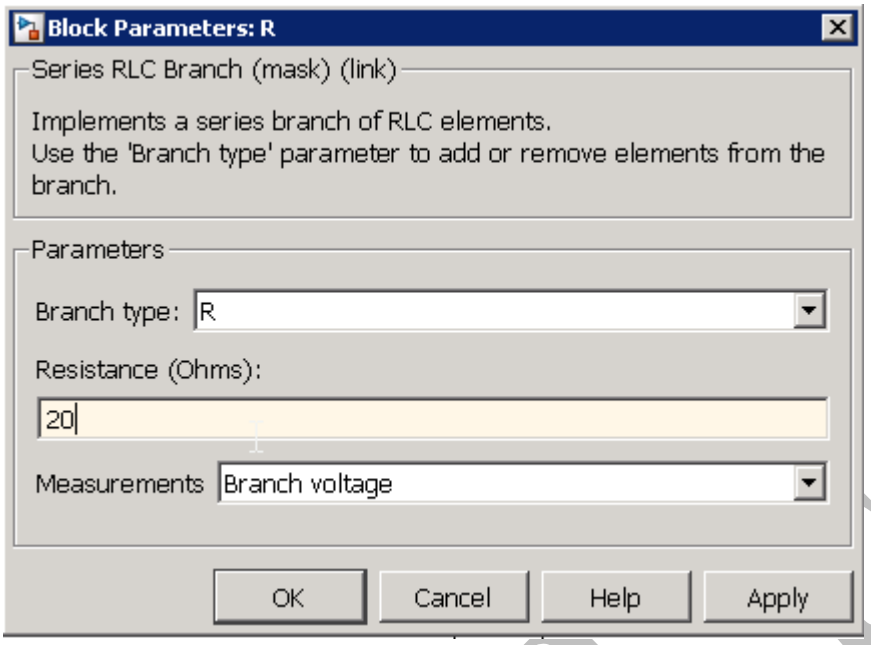
And for the intensity by double clicking on intensity measurement block:



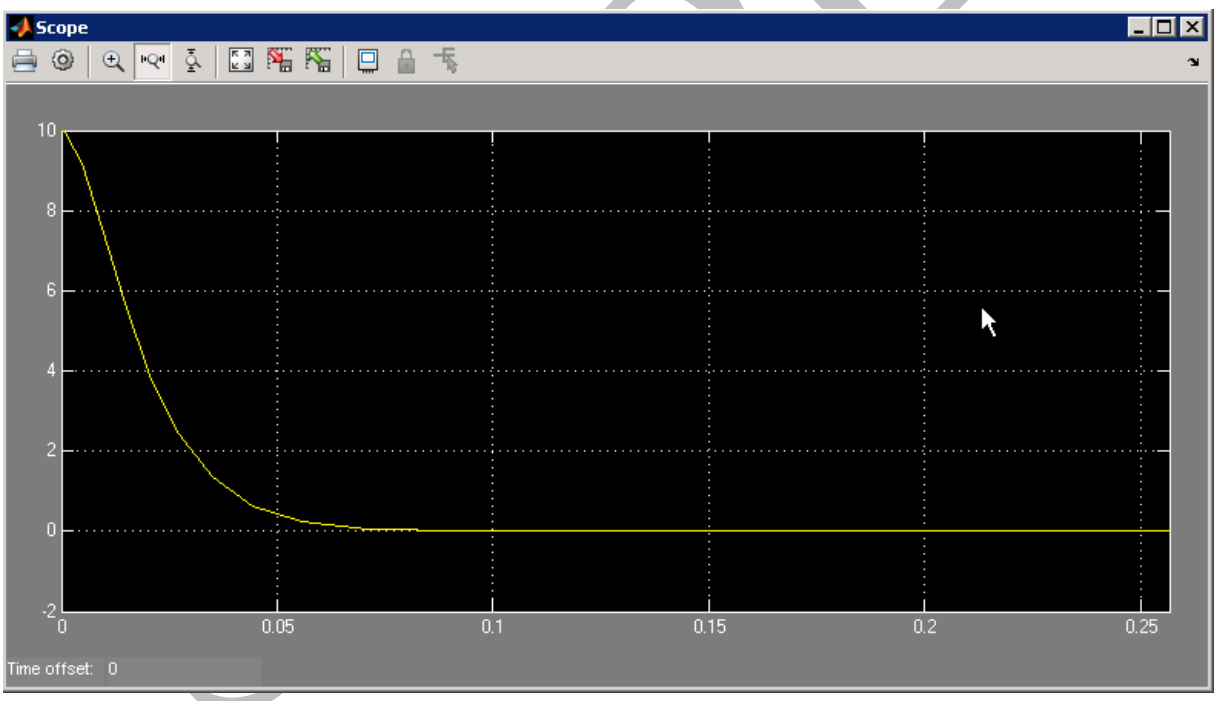
2. Critically over-damped circuit

For refresh this is the case where:

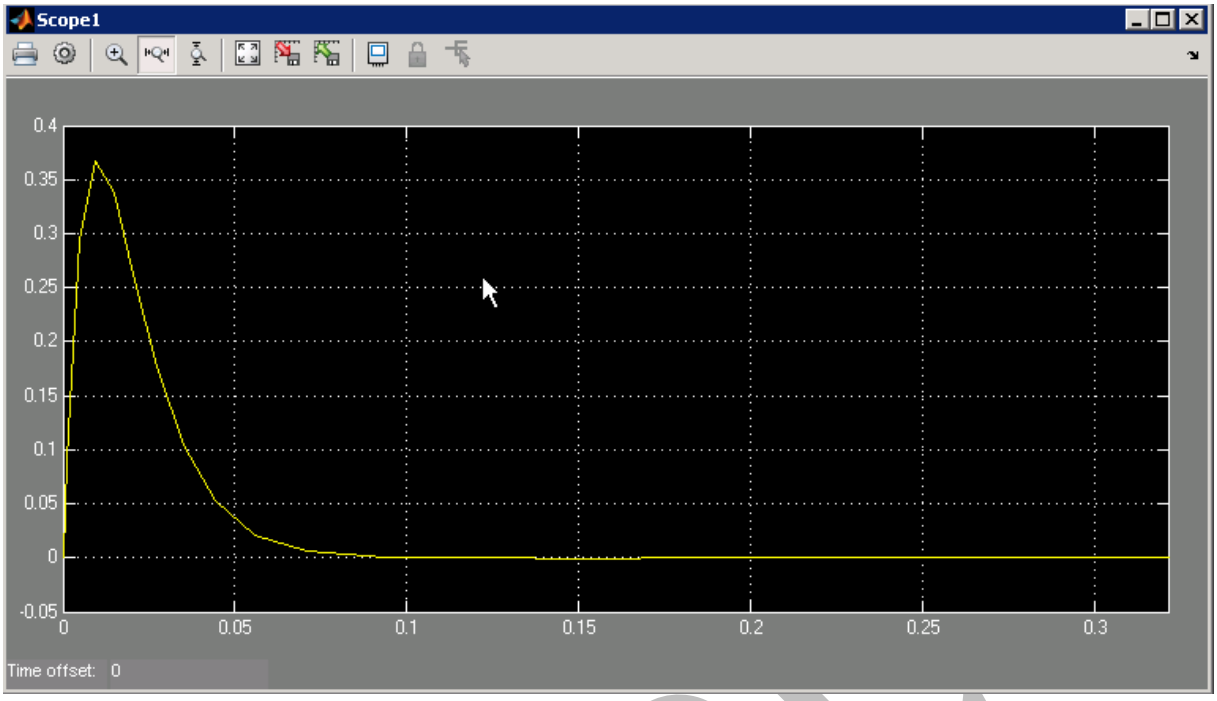
$$R = 2\sqrt{\frac{L}{C}}$$



When we run the simulation and we double click on the voltage measurement we get:



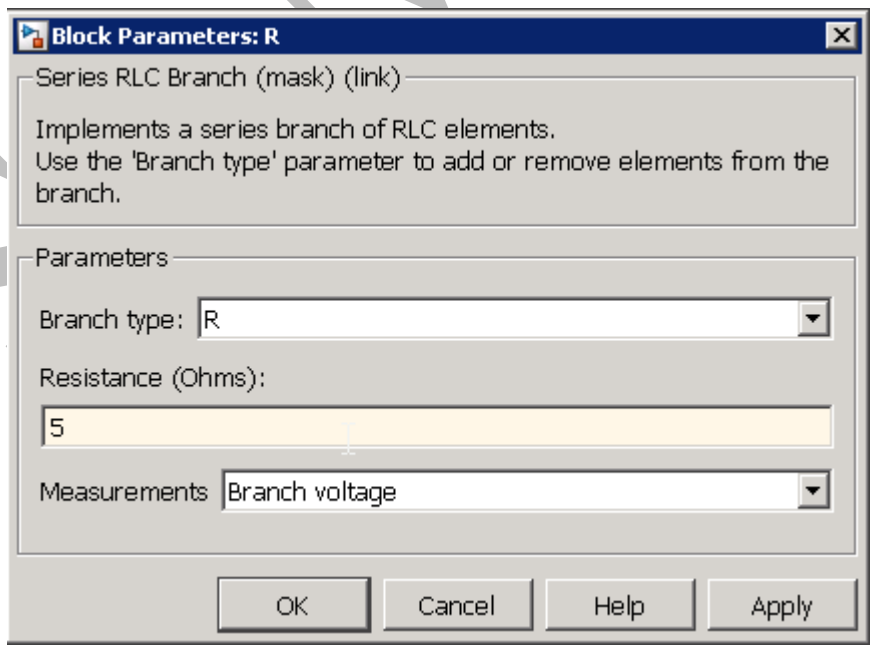
And for the intensity by double clicking on intensity measurement block:



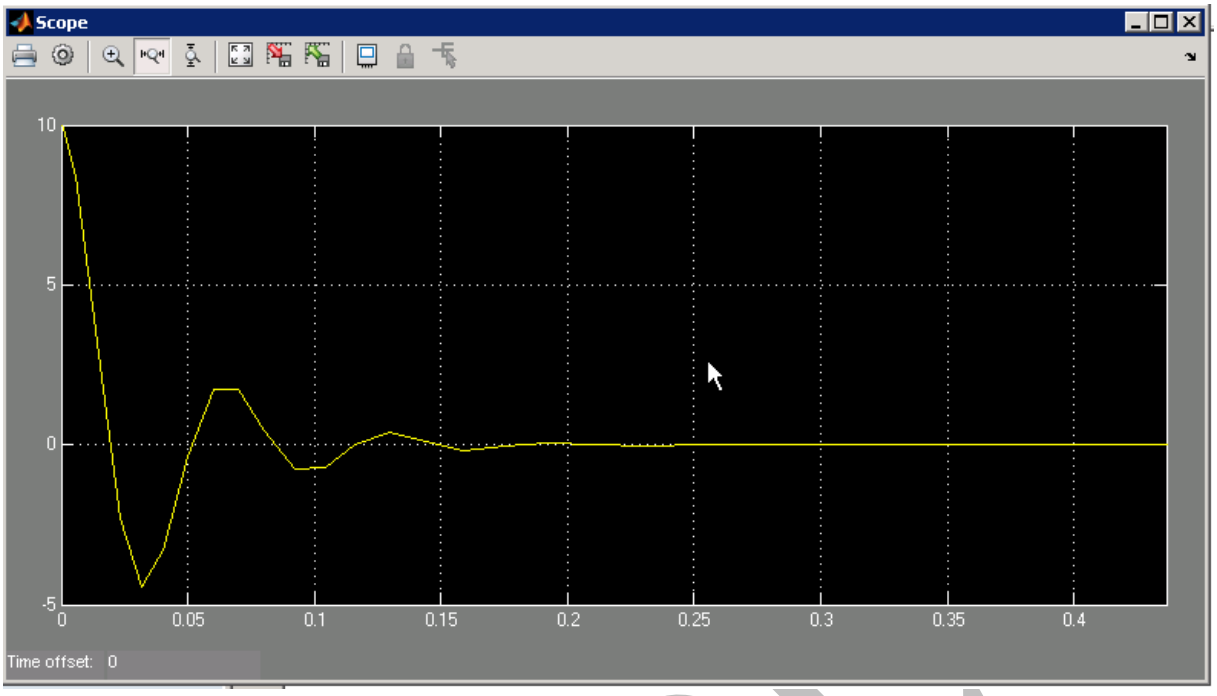
3. Under-damped circuit

For refresh this is the case where:

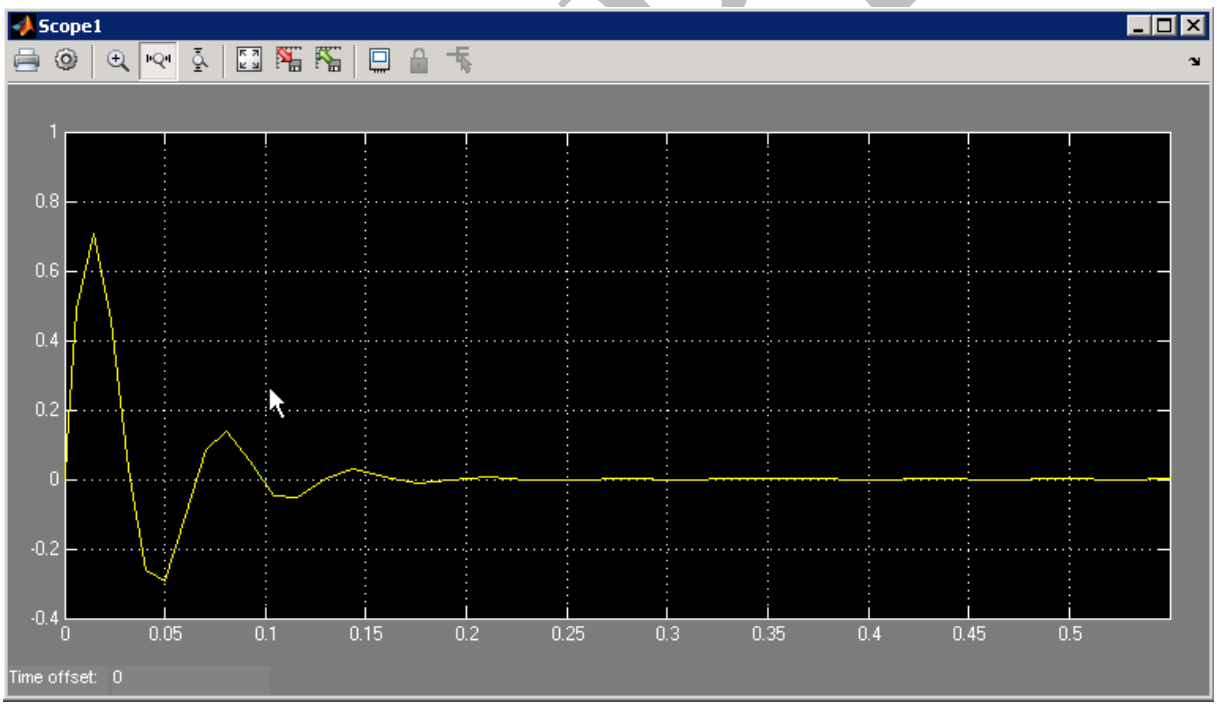
$$R < 2\sqrt{\frac{L}{C}}$$



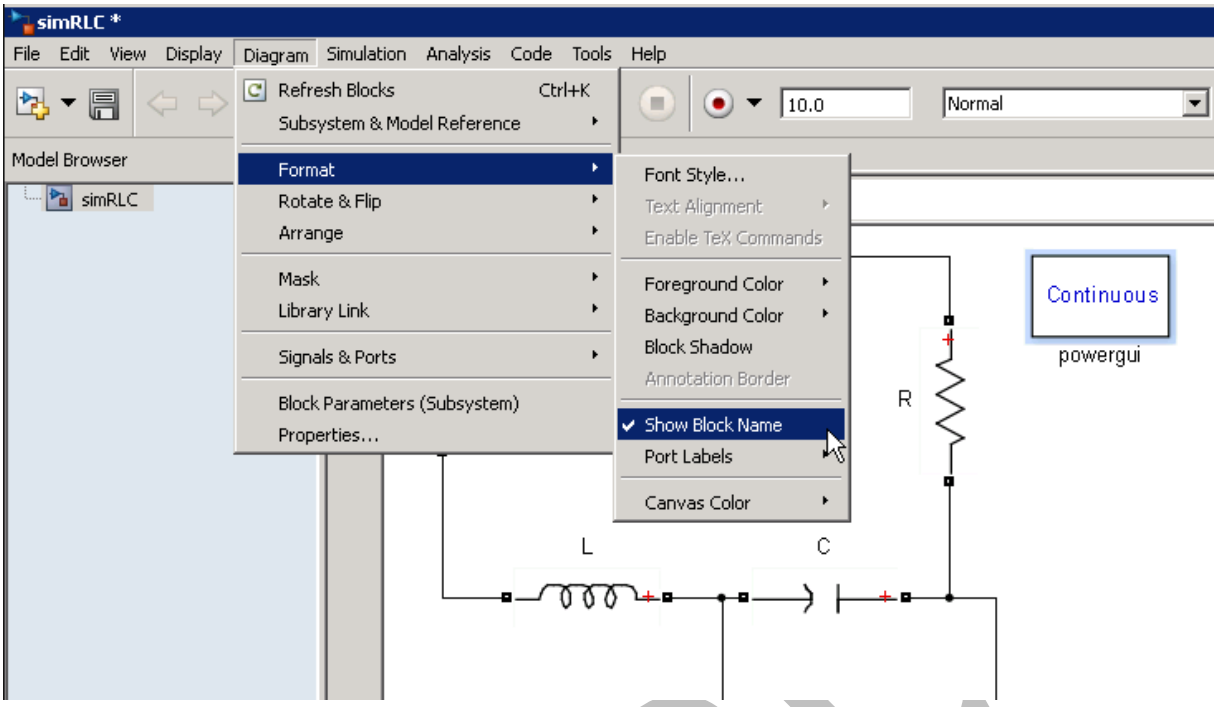
When we run the simulation and we double click on the voltage measurement we get:



And for the intensity by double clicking on intensity measurement block:



To finish, if you want to hide the names of the blocks you can use (don't hesitate to make multiple selection!):

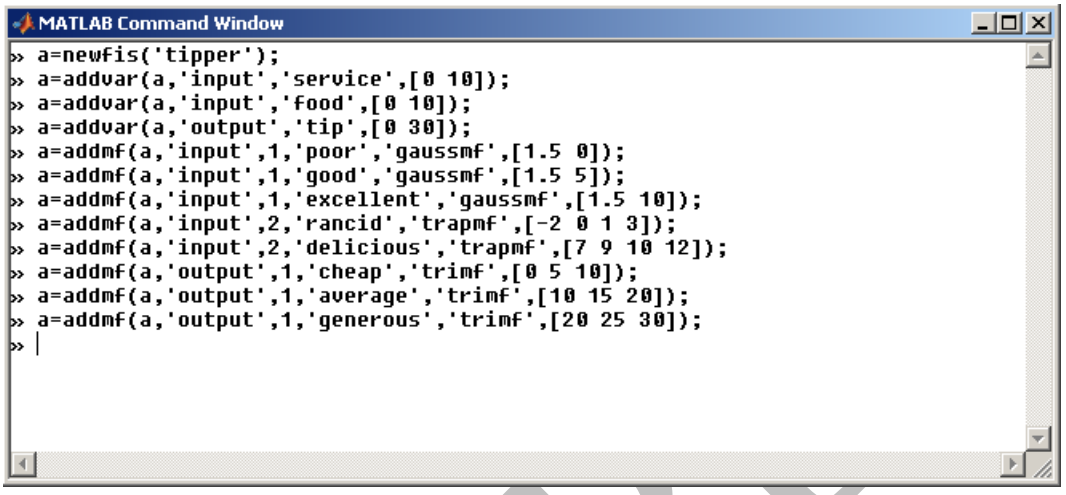

simRLC *
File Edit View Display Diagram Simulation Analysis Code Tools Help
Refresh Blocks Ctrl+K
Subsystem & Model Reference
Format
Rotate & Flip
Arrange
Mask
Library Link
Signals & Ports
Block Parameters (Subsystem)
Properties...
Font Style...
Text Alignment
Enable TeX Commands
Foreground Color
Background Color
Block Shadow
Annotation Border
Show Block Name
Port Labels
Canvas Color
10.0
Normal
Model Browser
simRLC
Continuous
powergui
R
L
C

SAMPLE

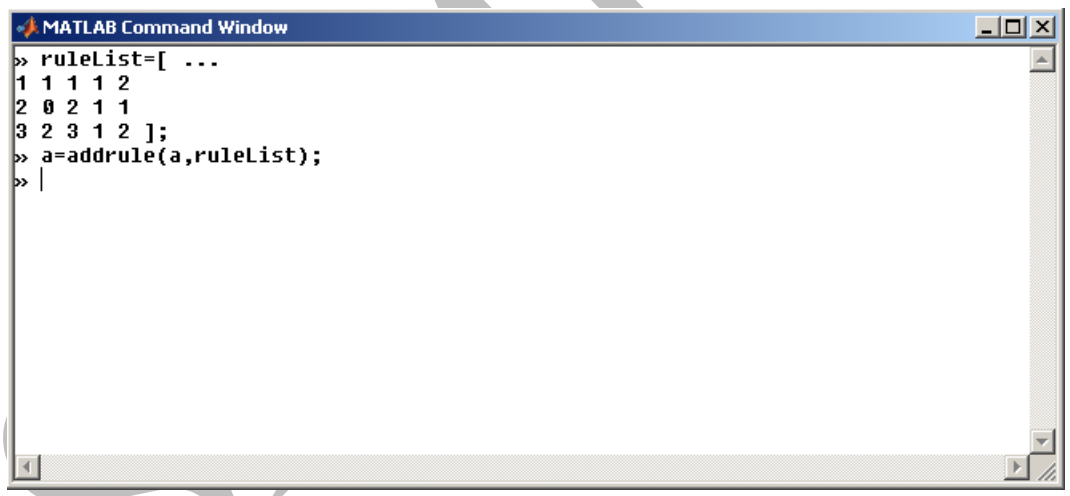
Fuzzy Logic (Fuzzy Toolbox)

The purpose here is to reproduce what we have spoken about in the theoretical training about fuzzy logic and taking into consideration the "tipper problem".

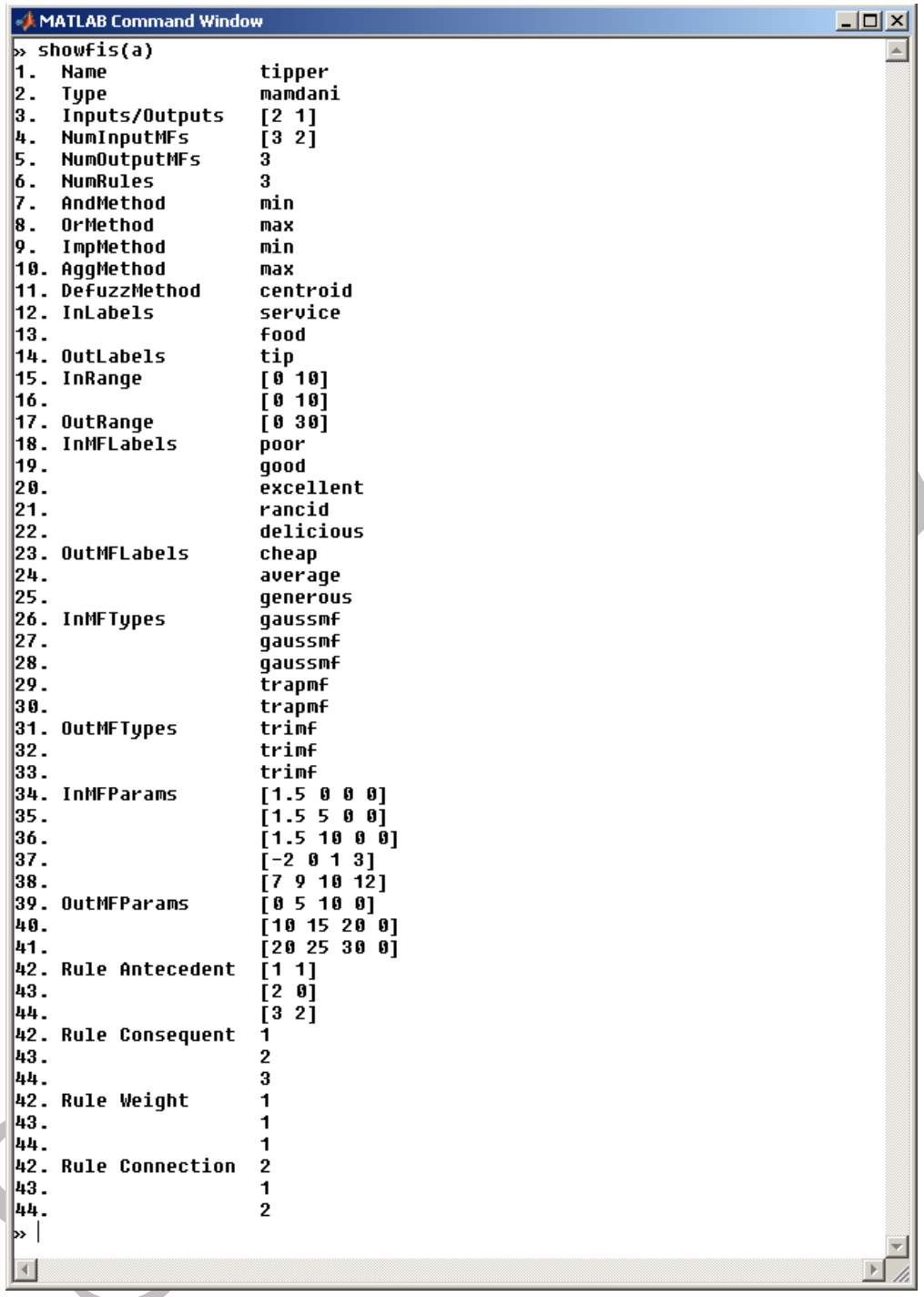
First we create the input and output variables:



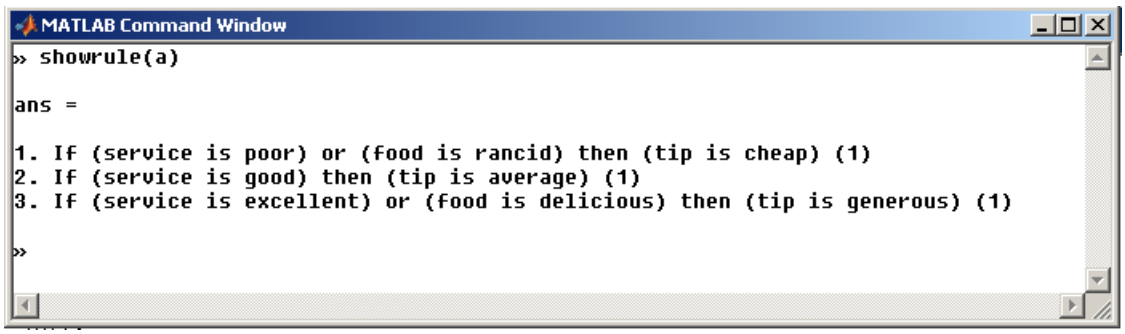
After we associate all those rules together:



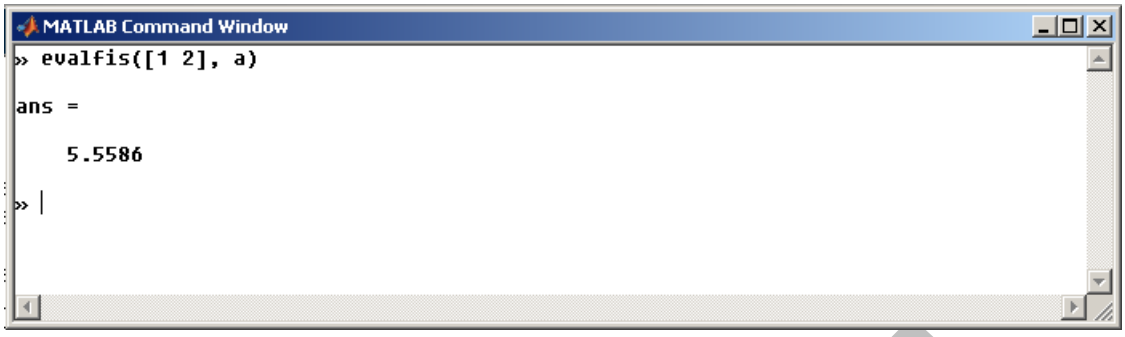
We can display a summary:



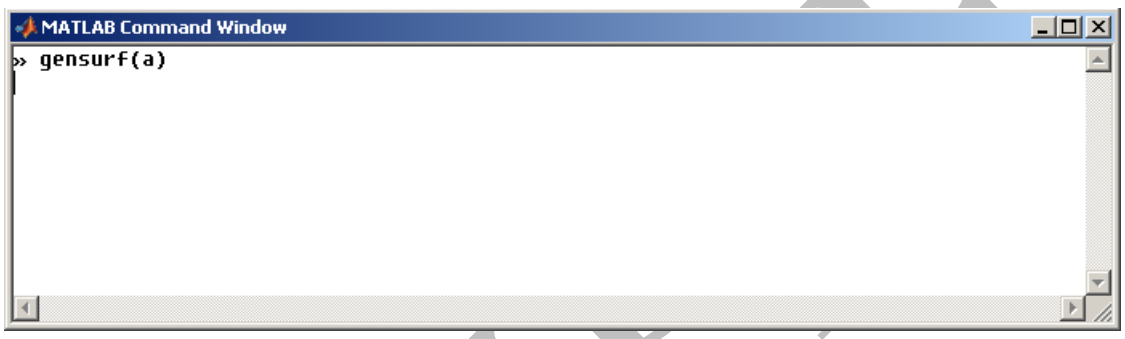
Or we can ask to show the rules explicitly:

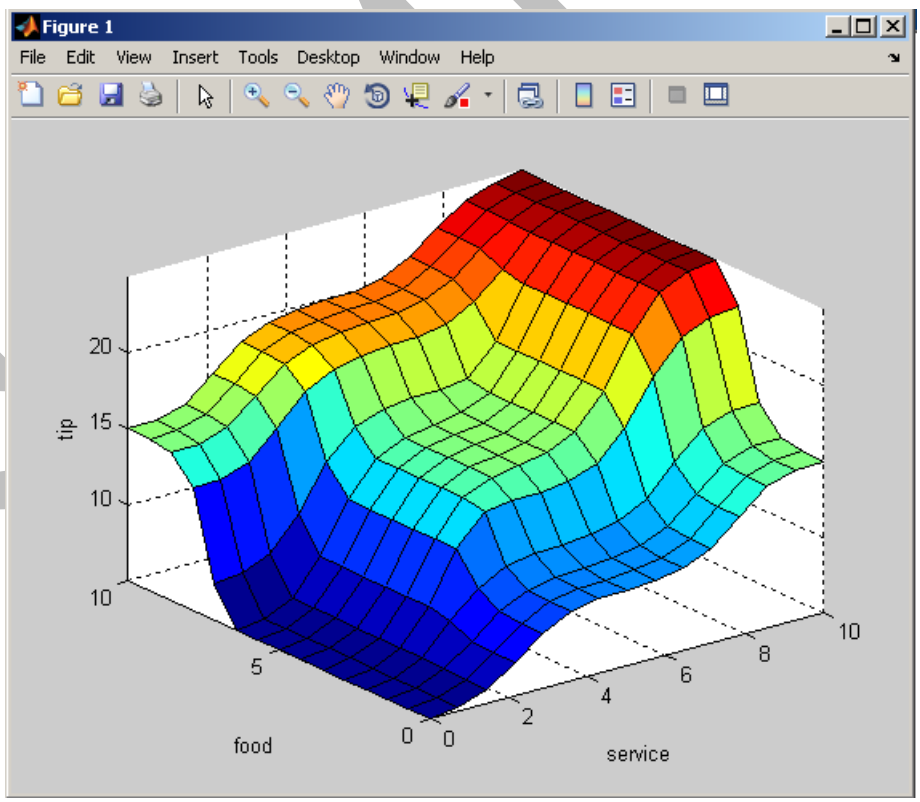


We can evaluate the output for a given output very quickly:



and finally to get the corresponding plot:





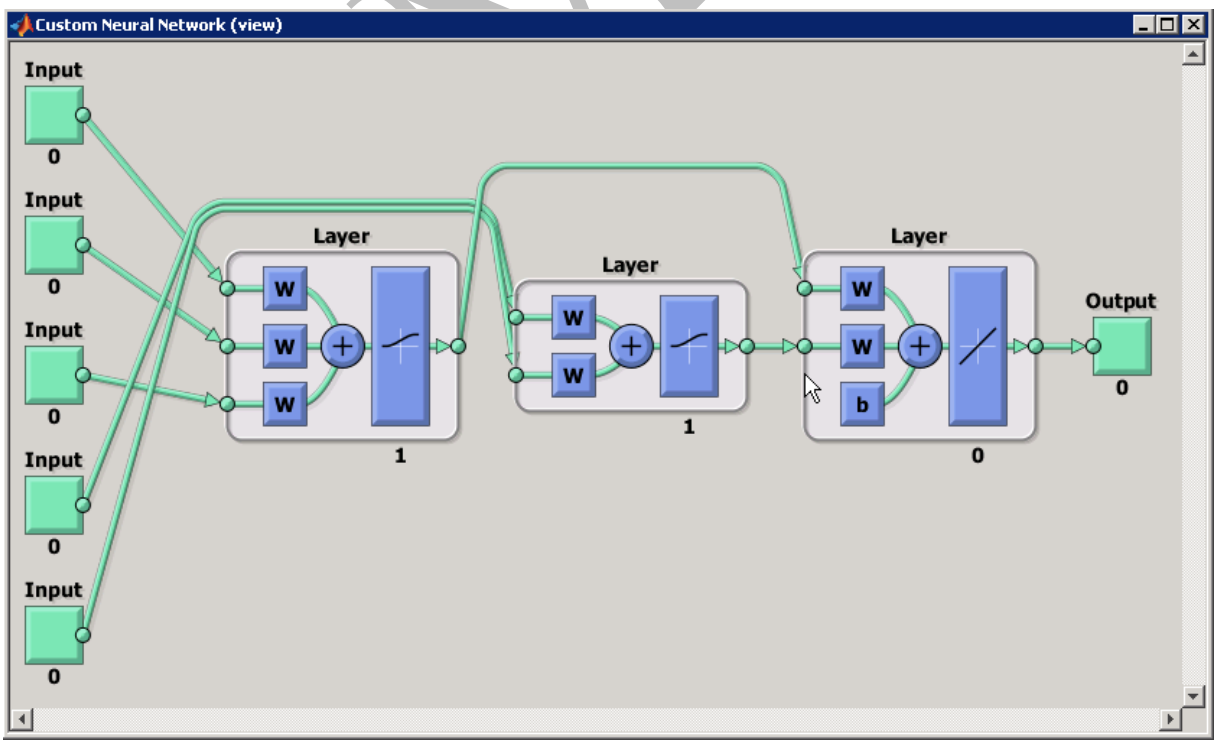
Neural Networks (Neural Network Toolbox)

http://www.MathWorks.ch/ch/help/nnet/release-notes.html

Multiple input/One output Neural Network (MIOO-NN)

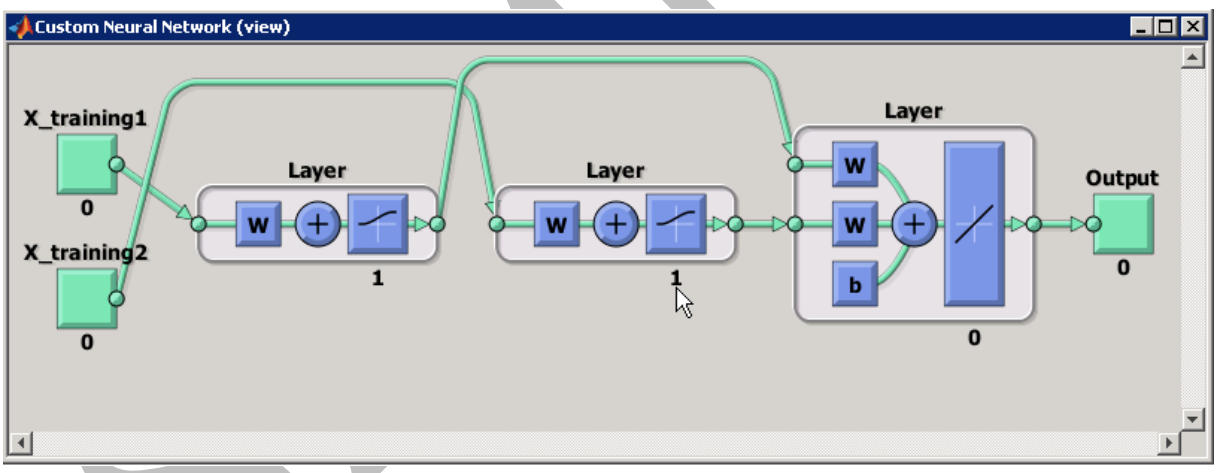
EDITOR PUBLISH VIEW

```
net=network(...
    5,... %number of inputs
    3,... %number of layers
    [0;0;1],... %bias input for layer i (third layer only in this case)
    [1 1 1 0 0;0 0 0 1 1;0 0 0 0 0],... %original inputs coming into the layer i
    [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
    [0 0 1]... %has the layer j an independant output?
);
%we choose the transfer function for each layer
%logsig, purelin, hardlim, logsig
net.layers{1}.transferFcn='logsig';
net.layers{2}.transferFcn='logsig';
%number of hidden layers in each layer
net.layers{1}.size=1;
net.layers{2}.size=1;
net.layers{3}.size=0;
view(net)
```



EDITOR PUBLISH VIEW

```
clear all;close all;clc
net=network(...
    2,... %number of inputs
    3,... %number of layers
    [0;0;1],... %bias input for layer i (third layer only in this case)
    [1 0;0 1;0 0],... %original inputs coming into the layer i
    [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
    [0 0 1]... %has the layer j an independant output?
);
%we choose the transfer function for each layer
%logsig, purelin, hardlim, logsig
net.layers{1}.transferFcn='logsig';
net.layers{2}.transferFcn='logsig';
%number of hidden layers in each layer
net.layers{1}.size=1;
net.layers{2}.size=1;
net.layers{3}.size=0;
net.inputs{1}.name='X_training1';
net.inputs{2}.name='X_training2';
net.outputs{1}.name='Y_training';
view(net);
```

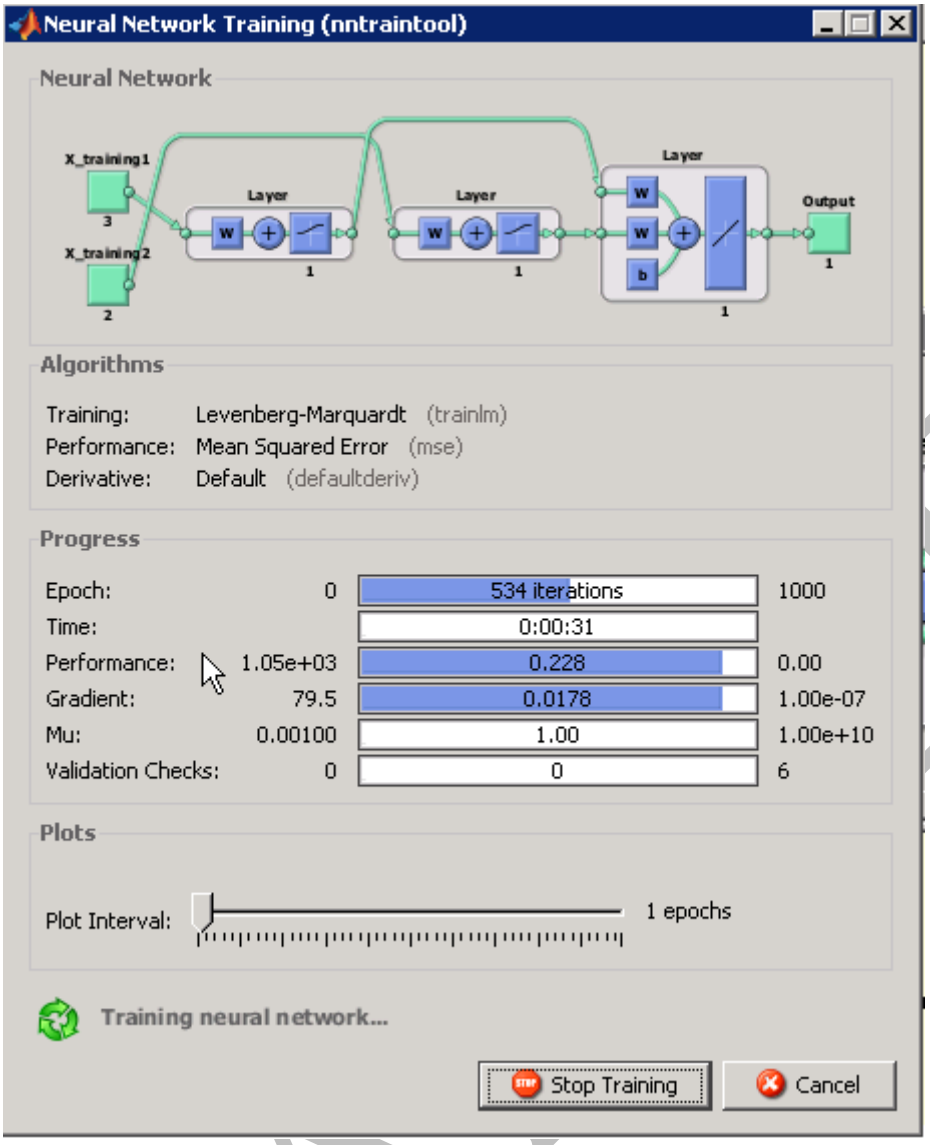


Now let us train the Neural Network:

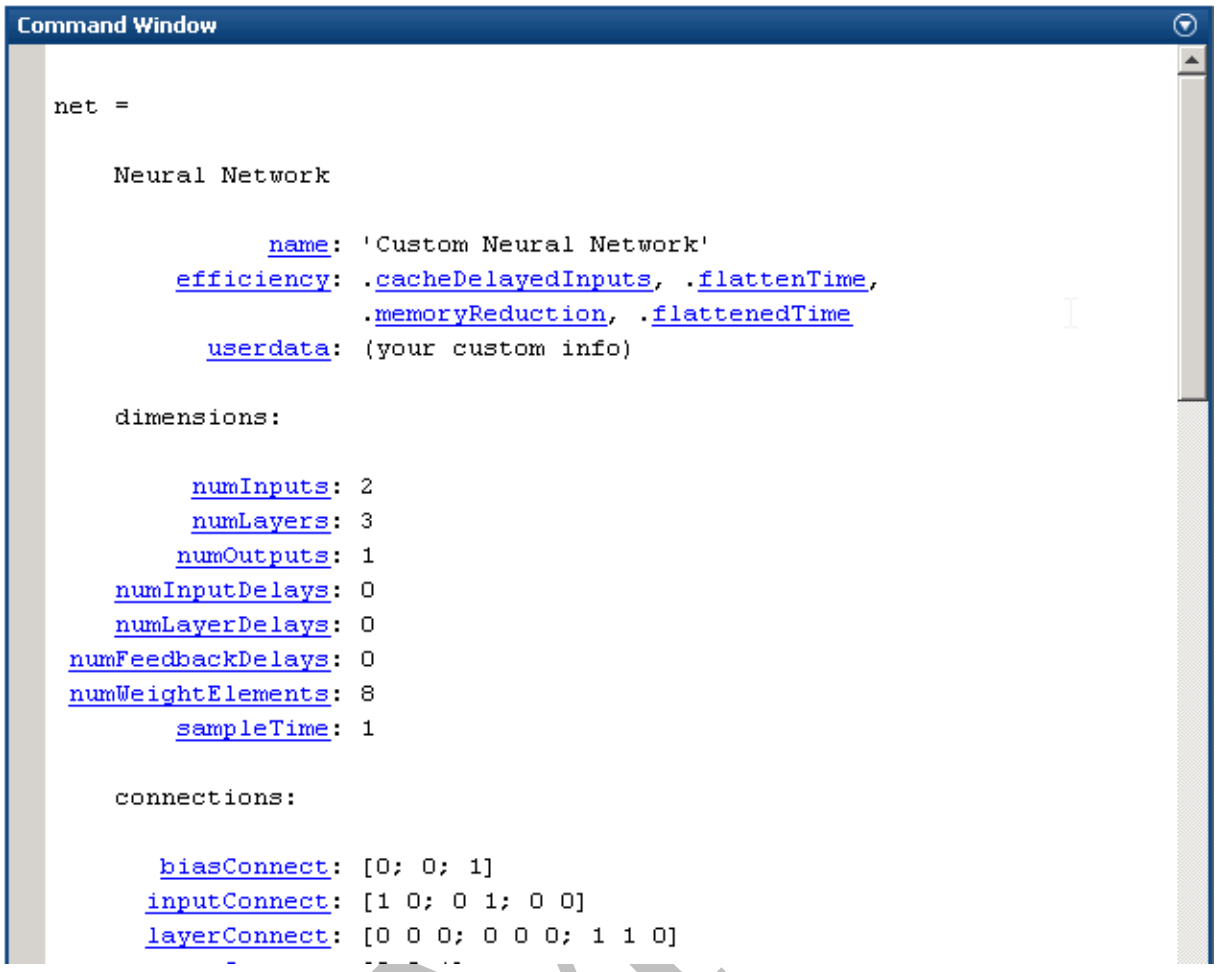
EDITOR PUBLISH VIEW

```
clear all;close all;clc
net=network(...
    2,... %number of inputs
    3,... %number of layers
    [0;0;1],... %bias input for layer i (third layer only in this case)
    [1 0;0 1;0 0],... %original inputs coming into the layer i
    [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
    [0 0 1]... %has the layer j an independant output?
);
%we choose the transfer function for each layer
%logsig, purelin, hardlim, logsig
net.layers{1}.transferFcn='logsig';
net.layers{2}.transferFcn='logsig';
%number of hidden layers in each layer
net.layers{1}.size=1;
net.layers{2}.size=1;
net.layers{3}.size=0;
net.inputs{1}.name='X_training1';
net.inputs{2}.name='X_training2';
net.outputs{1}.name='Y_training';
net.numinputs = 2;
view(net);

X_training=double(dataset('XLSFile','c:\tmp\nn\NeuralNetworkTrainingSet.xls'));
X_training1=X_training(:,1:3);
X_training2=X_training(:,4:5);
Y_training=double(dataset('XLSFile','c:\tmp\nn\NeuralNetworkTrainingTarget.xls'));
net.trainFcn='trainlm';
net.performFcn='mse';
net=train(net,{X_training1';X_training2'},Y_training')
```


Neural Network Training (nntraintool)
Neural Network
X_training1
3
X_training2
2
Layer
1
Layer
1
Layer
1
Output
1
Algorithms
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)
Progress
Epoch: 0 534 iterations 1000
Time: 0:00:31
Performance: 1.05e+03 0.228 0.00
Gradient: 79.5 0.0178 1.00e-07
Mu: 0.00100 1.00 1.00e+10
Validation Checks: 0 0 6
Plots
Plot Interval: 1 epochs
Training neural network...
Stop Training
Cancel

SAMPLE


Command Window
net =
Neural Network
name: 'Custom Neural Network'
efficiency: .cacheDelayedInputs, .flattenTime,
.memoryReduction, .flattenedTime
userdata: (your custom info)
dimensions:
numInputs: 2
numLayers: 3
numOutputs: 1
numInputDelays: 0
numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 8
sampleTime: 1
connections:
biasConnect: [0; 0; 1]
inputConnect: [1 0; 0 1; 0 0]
layerConnect: [0 0 0; 0 0 0; 1 1 0]

SAM

```
    outputConnect: [0 0 1]

subobjects:

          inputs: {2x1 cell array of 2 inputs}
          layers: {3x1 cell array of 3 layers}
         outputs: {1x3 cell array of 1 output}
          biases: {3x1 cell array of 1 bias}
    inputWeights: {3x2 cell array of 2 weights}
    layerWeights: {3x3 cell array of 2 weights}

functions:

        adaptFcn: (none)
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: (none)
     divideParam: (none)
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {}
      plotParams: {1x0 cell array of 0 params}
        trainFcn: 'trainlm'
      trainParam: .showWindow, .showCommandLine, .show, .epochs,
                  .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                  .mu_inc, .mu_max
```

```
weight and bias values:

              IW: {3x2 cell} containing 2 input weight matrices
              LW: {3x3 cell} containing 2 layer weight matrices
               b: {3x1 cell} containing 1 bias vector

methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

evaluate:       outputs = net(inputs)
```

```
Command Window
>> net.IW

ans =

    [1x3 double]                []
                []     [1x2 double]
                []                []

>> net.IW{1}

ans =

   -0.0480    0.0673   -0.0585

>> net.IW{2,2}

ans =

    0.0532   -0.0078

>> net.LW

ans =

            []            []     []
            []            []     []
    [10.4258]     [10.7986]     []
```

```
>> net.b

ans =

    []
    []
    [20.2050]
```

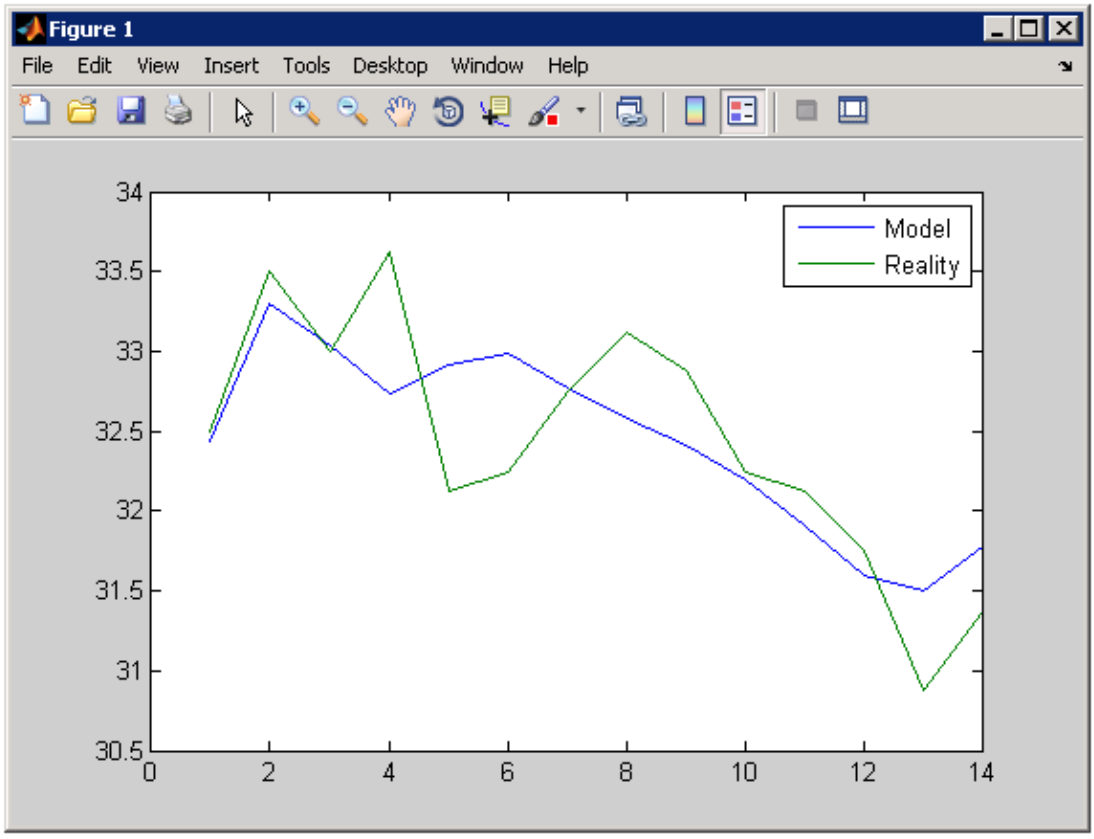
```
Command Window
>> model=[cell2mat(net({X_training1';X_training2'}))',Y_training]

model =

   32.4325   32.5000
   33.3047   33.5000
   33.0391   33.0000
   32.7406   33.6250
   32.9124   32.1250
   32.9916   32.2500
   32.7703   32.7500
   32.5749   33.1250
   32.4162   32.8750
   32.1965   32.2500
   31.9064   32.1250
   31.5933   31.7500
   31.5004   30.8750
   31.7761   31.3750

fx >>
```

```
Command Window
>> plot(1:14,model(:,1),1:14,model(:,2))
>> legend('Model','Reality')
fx >>
```



```
Command Window
>> SSE=sum((model(:,1)-model(:,2)).^2)

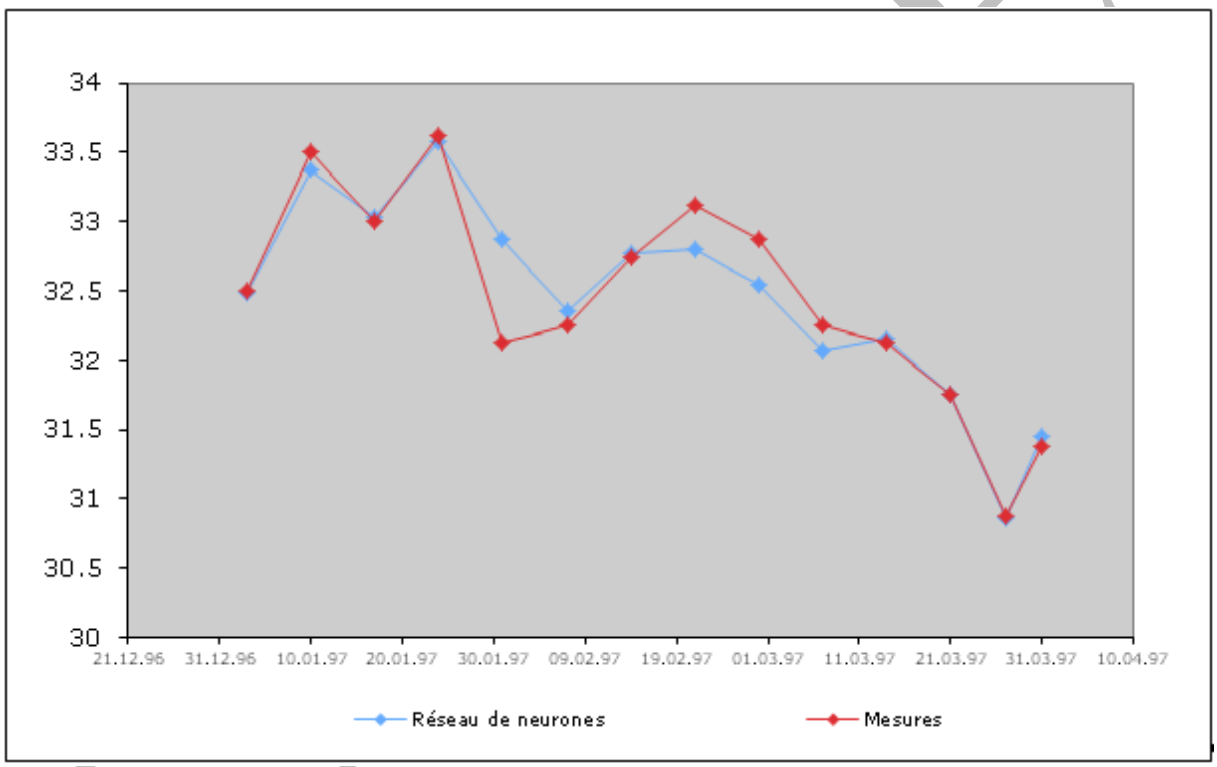
SSE =

    3.1371

fx >>
```

Worse than with Microsoft Excel...

To compare with Microsoft Excel:



EDITOR PUBLISH VIEW

```
43      %----------------------------------
44      %and what about back-testing?
45 -    X_test1=[8.25 8 8.13 8 8.63 9.25;...
46      39.13 37.63 38.88 39.13 41.38 42.25;...
47      51 50.25 52.25 51.63 53.38 53.88];
48 -    X_test2=[30.13 28.88 30.38 29.25 29.75 31.25;...
49      54 53 56.25 54.88 57 57.88];
50 -    Y_test=[30.88 32.25 34.25 34.25 34.75 36.63];
51 -    model=[cell2mat(net({X_test1;X_test2}))',Y_test'];
52 -    plot(1:6,model(:,1),1:6,model(:,2));
53 -    legend('Model','Reality');
54 -    SSE=sum((model(:,1)-model(:,2)).^2)
```

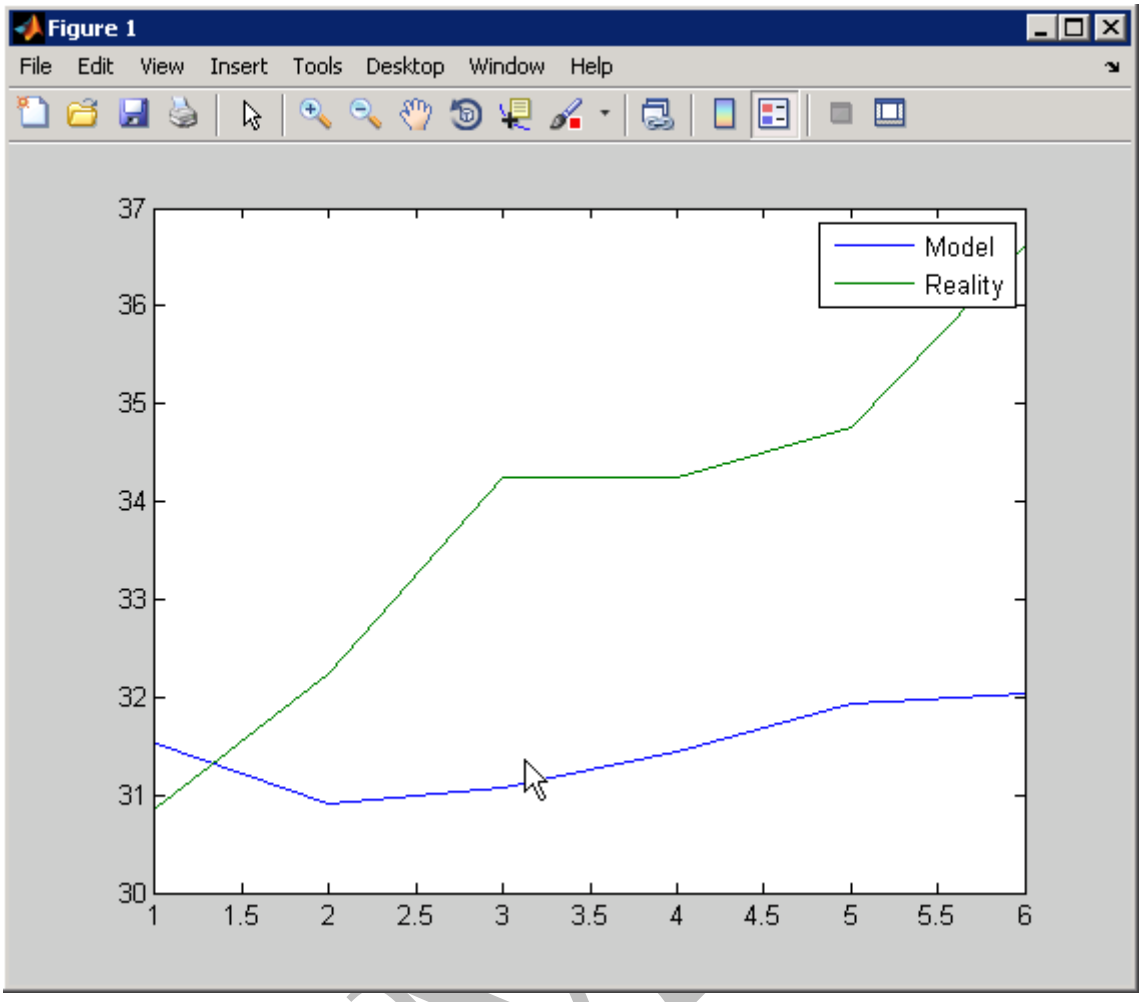
Command Window

```
>> X_test1=[8.25 8 8.13 8 8.63 9.25;...
39.13 37.63 38.88 39.13 41.38 42.25;...
51 50.25 52.25 51.63 53.38 53.88];
X_test2=[30.13 28.88 30.38 29.25 29.75 31.25;...
54 53 56.25 54.88 57 57.88];
Y_test=[30.88 32.25 34.25 34.25 34.75 36.63];
model=[cell2mat(net({X_test1;X_test2}))',Y_test'];
plot(1:6,model(:,1),1:6,model(:,2));
legend('Model','Reality');
SSE=sum((model(:,1)-model(:,2)).^2)

SSE =

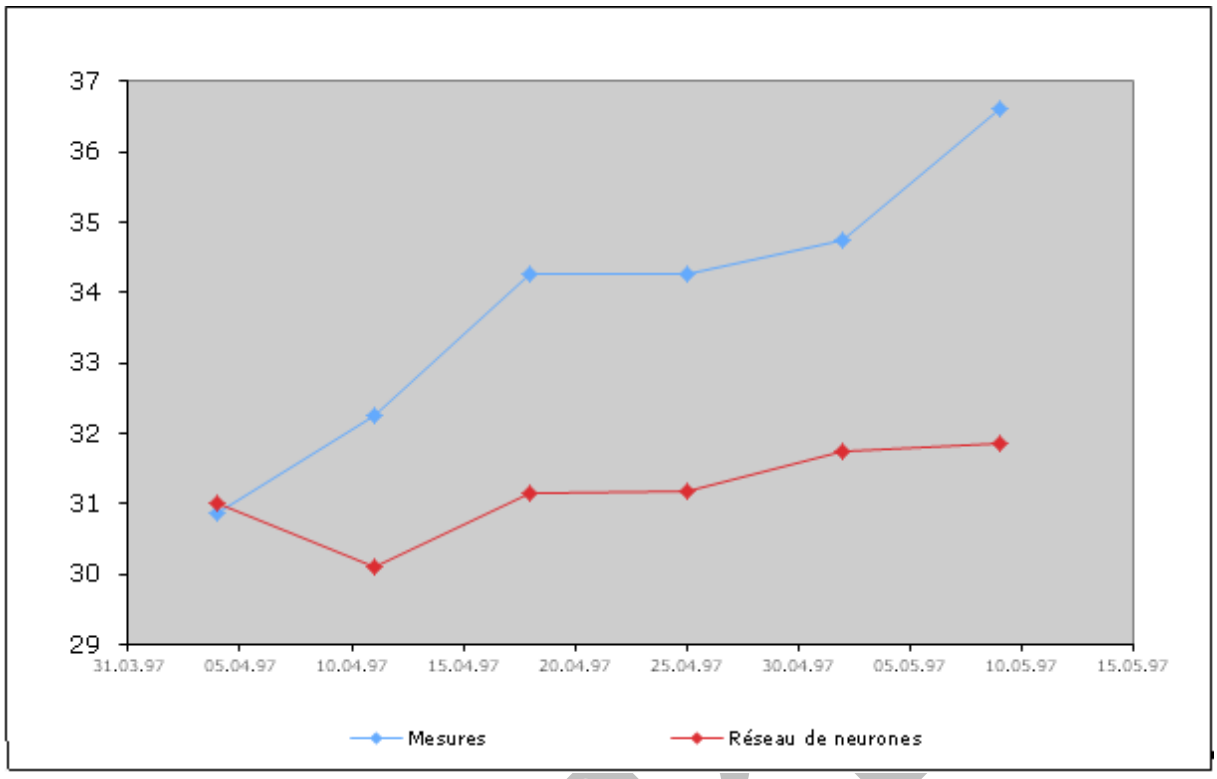
   49.0972

fx >>
```



And it Microsoft Excel we get:

SAM

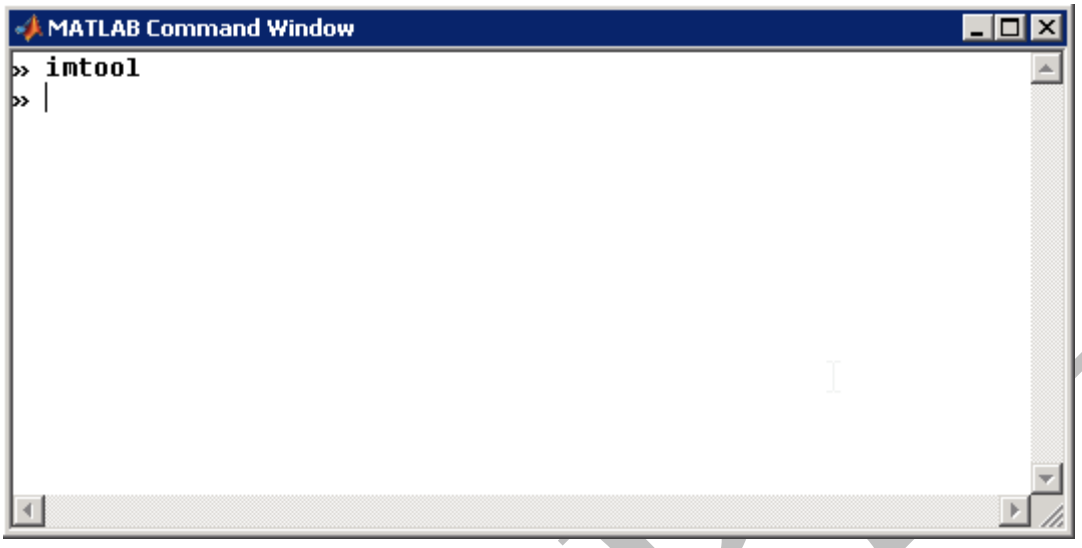


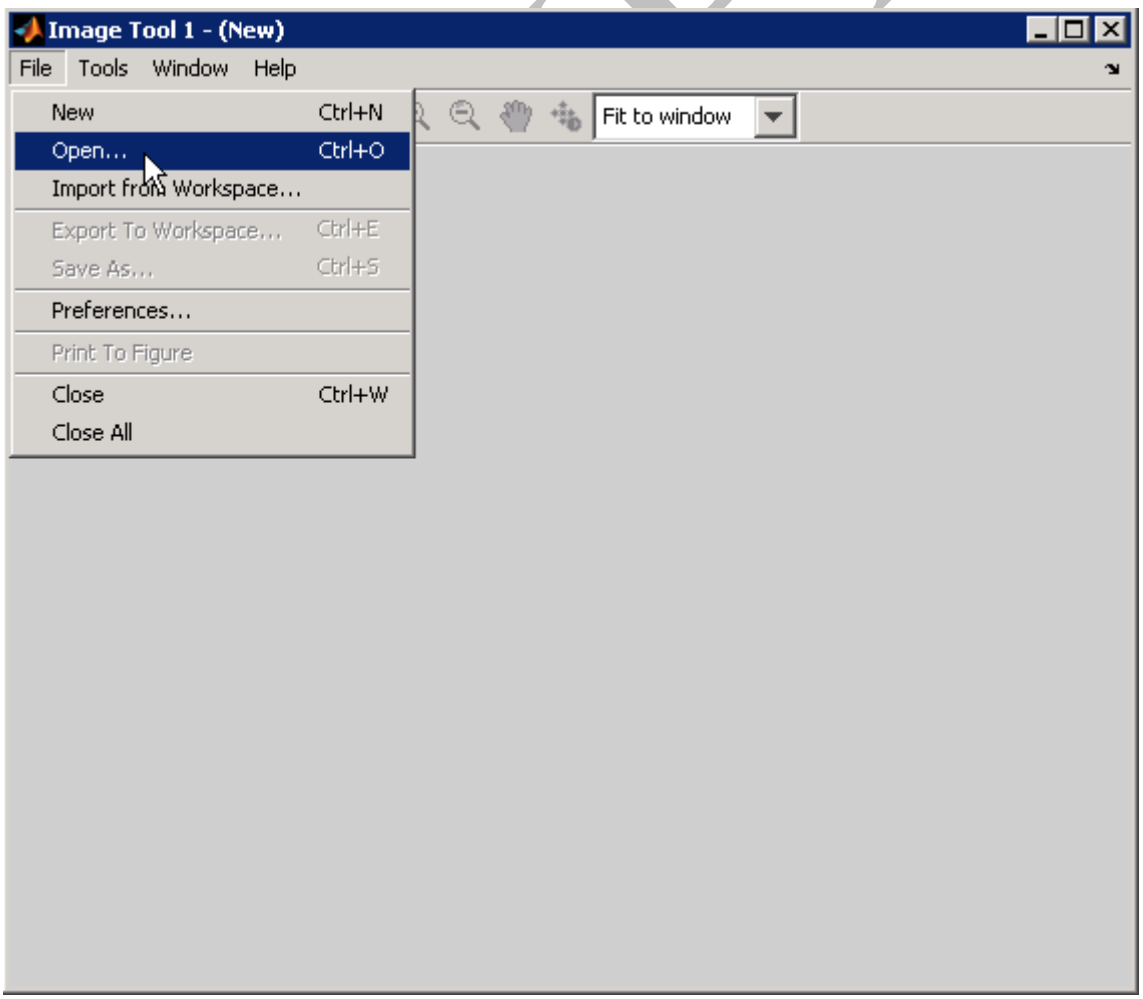
with a SSE of 55. MATLAB makes better for prediction!

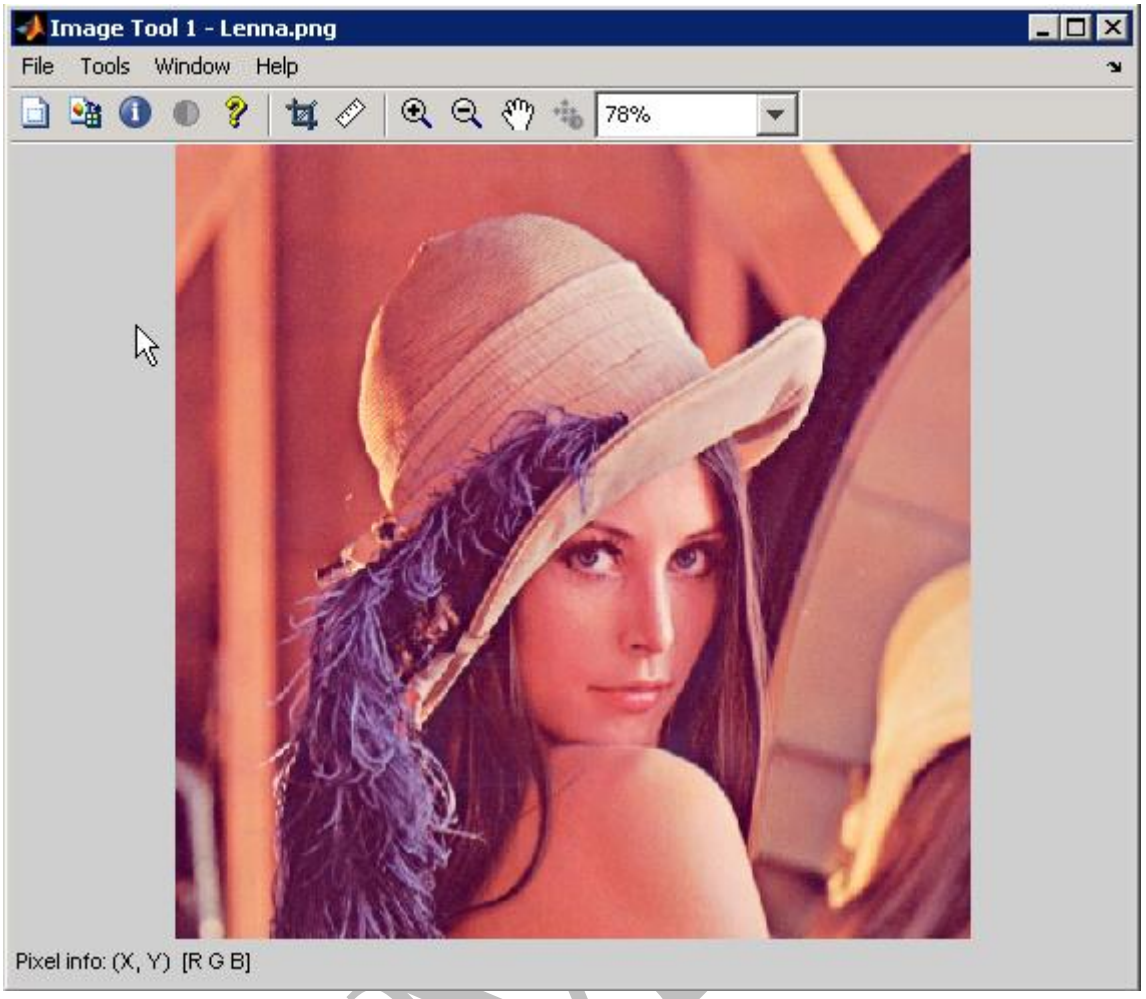
SAMPLE

Image Processing (with Image Processing and Acquisition+Statistical Toolboxes)

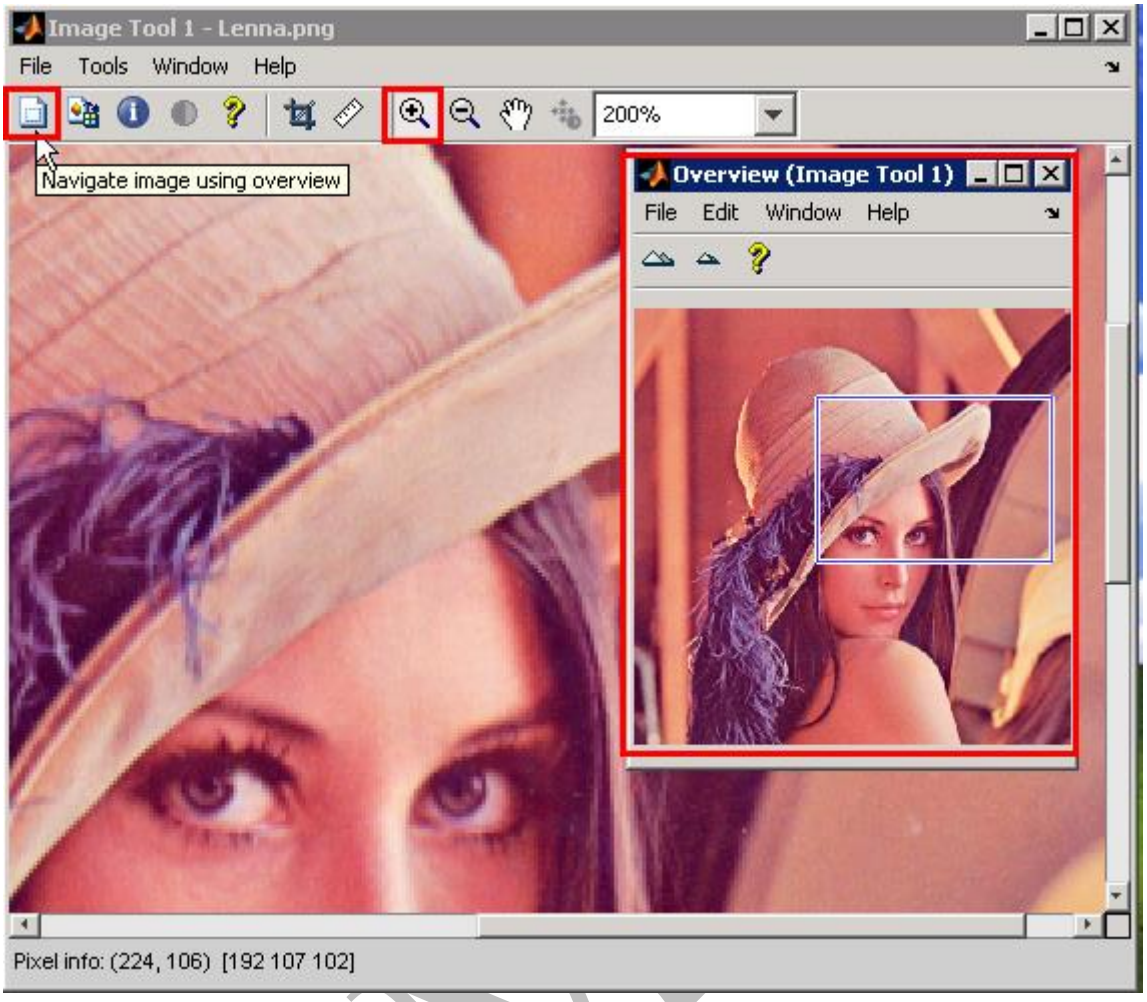
Play with the image tool wizard



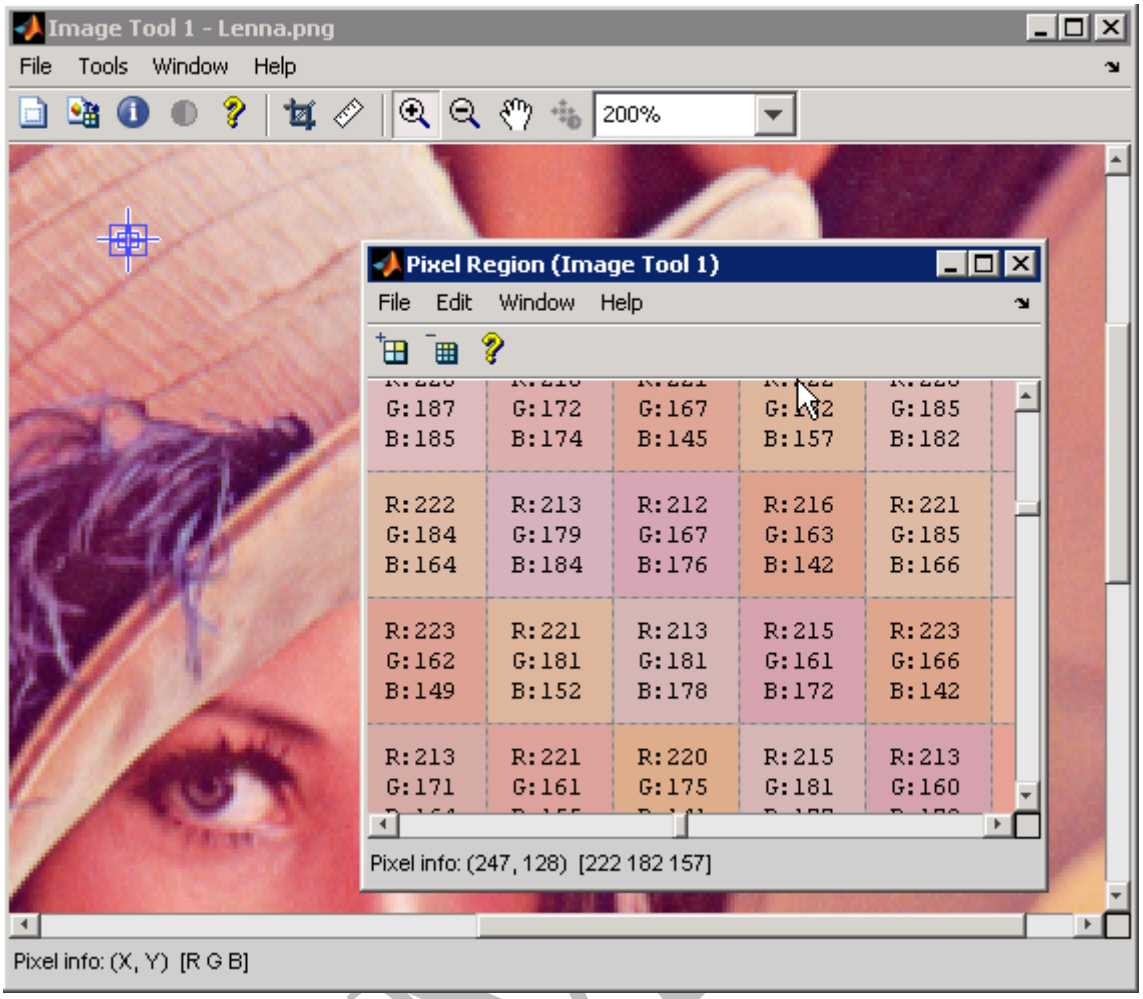



Image Tool 1 - Lenna.png
File Tools Window Help
78%
Pixel info: (X, Y) [R G B]

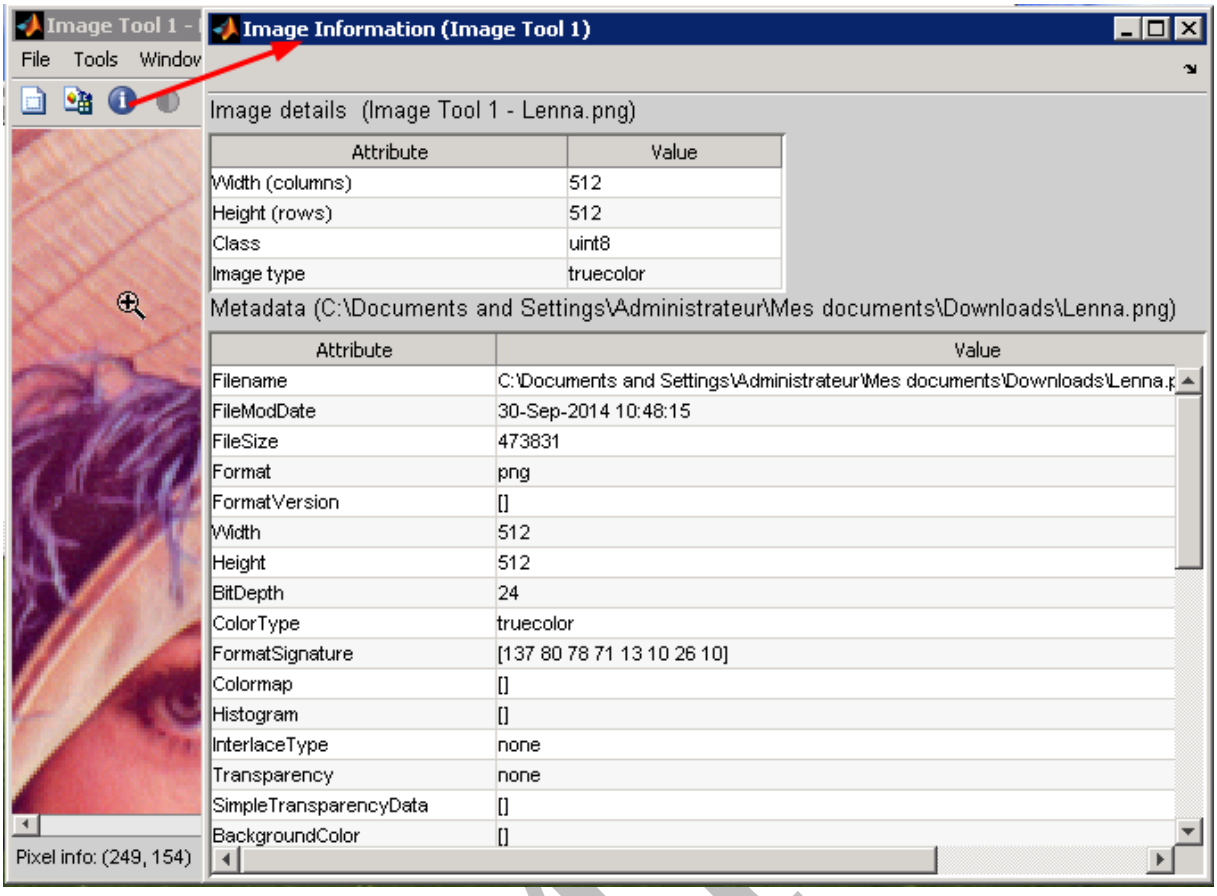
SAM


Image Tool 1 - Lenna.png
File Tools Window Help
200%
Navigate image using overview
Overview (Image Tool 1)
File Edit Window Help
Pixel info: (224, 106) [192 107 102]

SAM

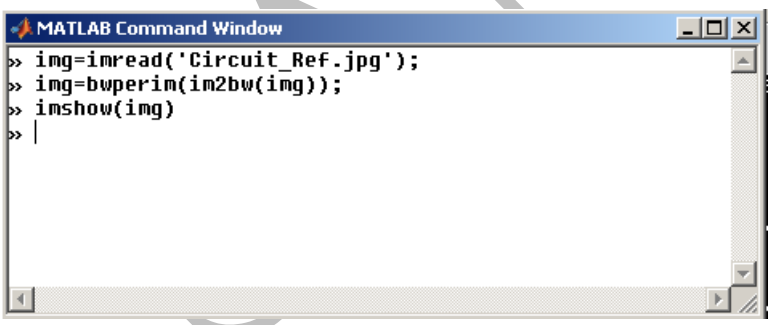

Image Tool 1 - Lenna.png
File Tools Window Help
200%
Pixel Region (Image Tool 1)
File Edit Window Help
G:187 G:172 G:167 G:185
B:185 B:174 B:145 B:157 B:182
R:222 R:213 R:212 R:216 R:221
G:184 G:179 G:167 G:163 G:185
B:164 B:184 B:176 B:142 B:166
R:223 R:221 R:213 R:215 R:223
G:162 G:181 G:181 G:161 G:166
B:149 B:152 B:178 B:172 B:142
R:213 R:221 R:220 R:215 R:213
G:171 G:161 G:175 G:181 G:160
Pixel info: (247, 128) [222 182 157]
Pixel info: (X, Y) [R G B]

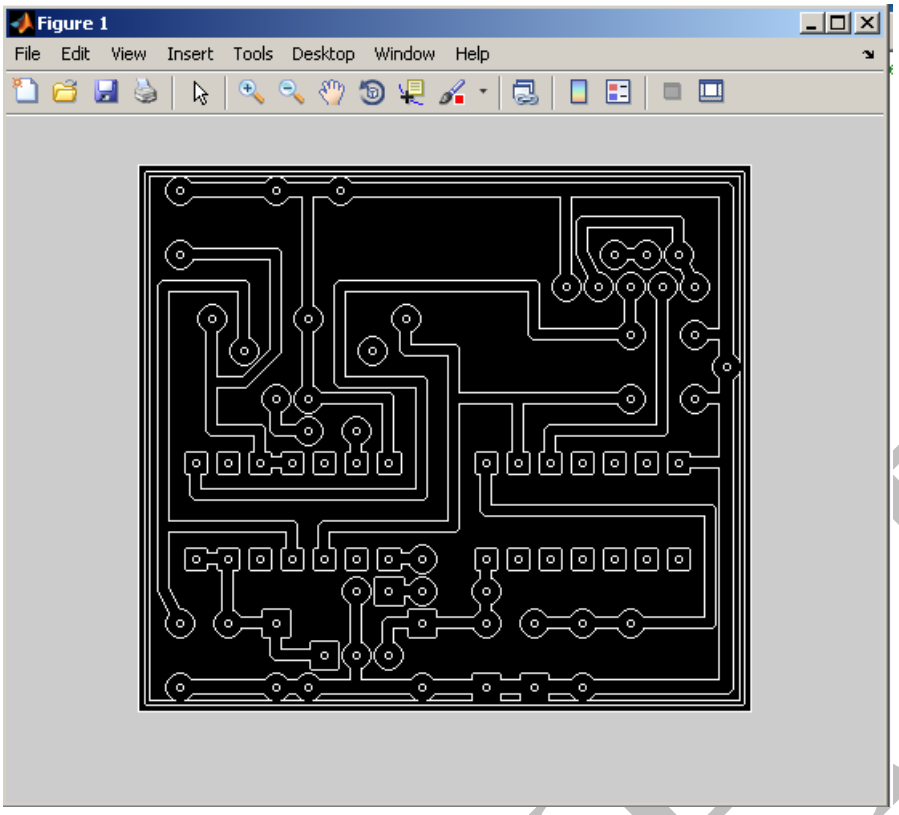
SAM



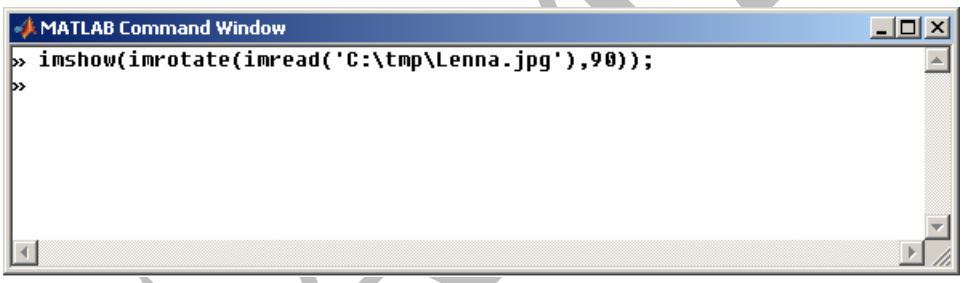
Convert image to black and white

This is especially useful for some MATLAB functions as for example the bwperim( ) whose purpose is to make perimeter detection on black&white image:

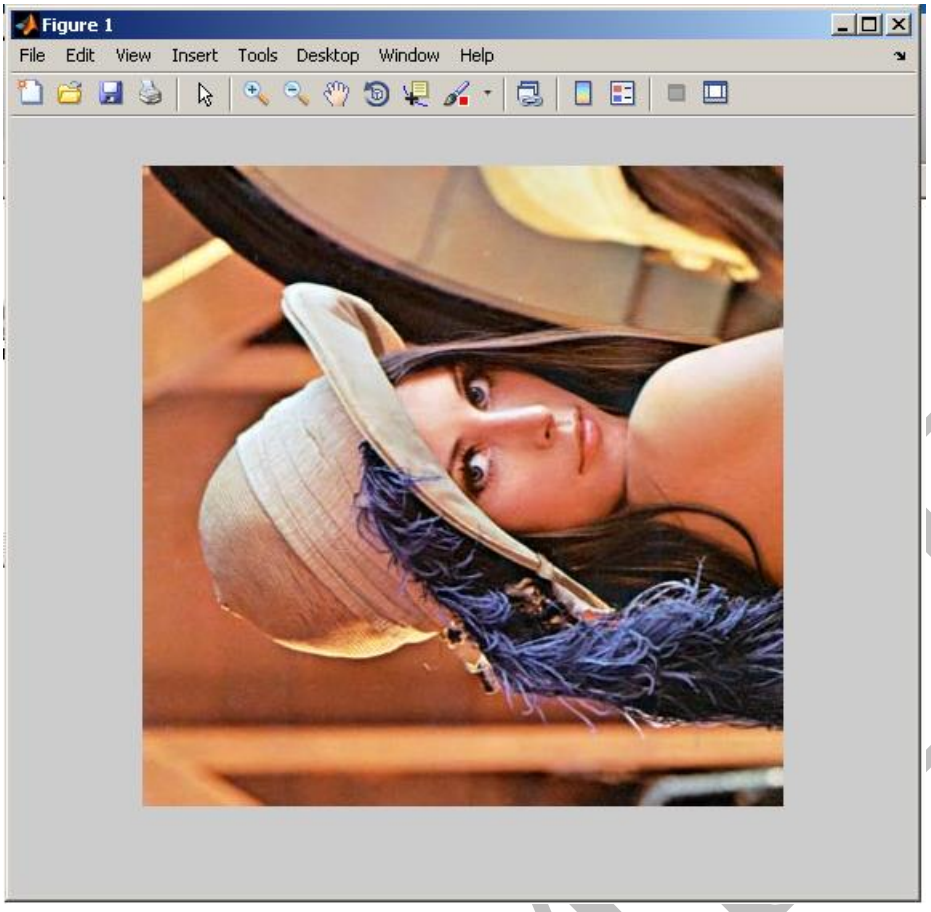




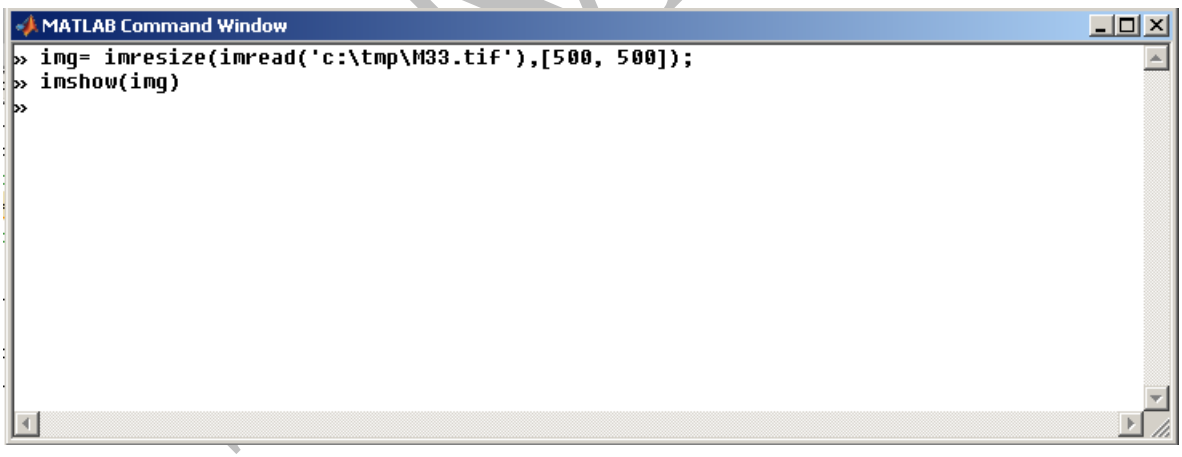
Rotate image



SAMPLE



Resize an image



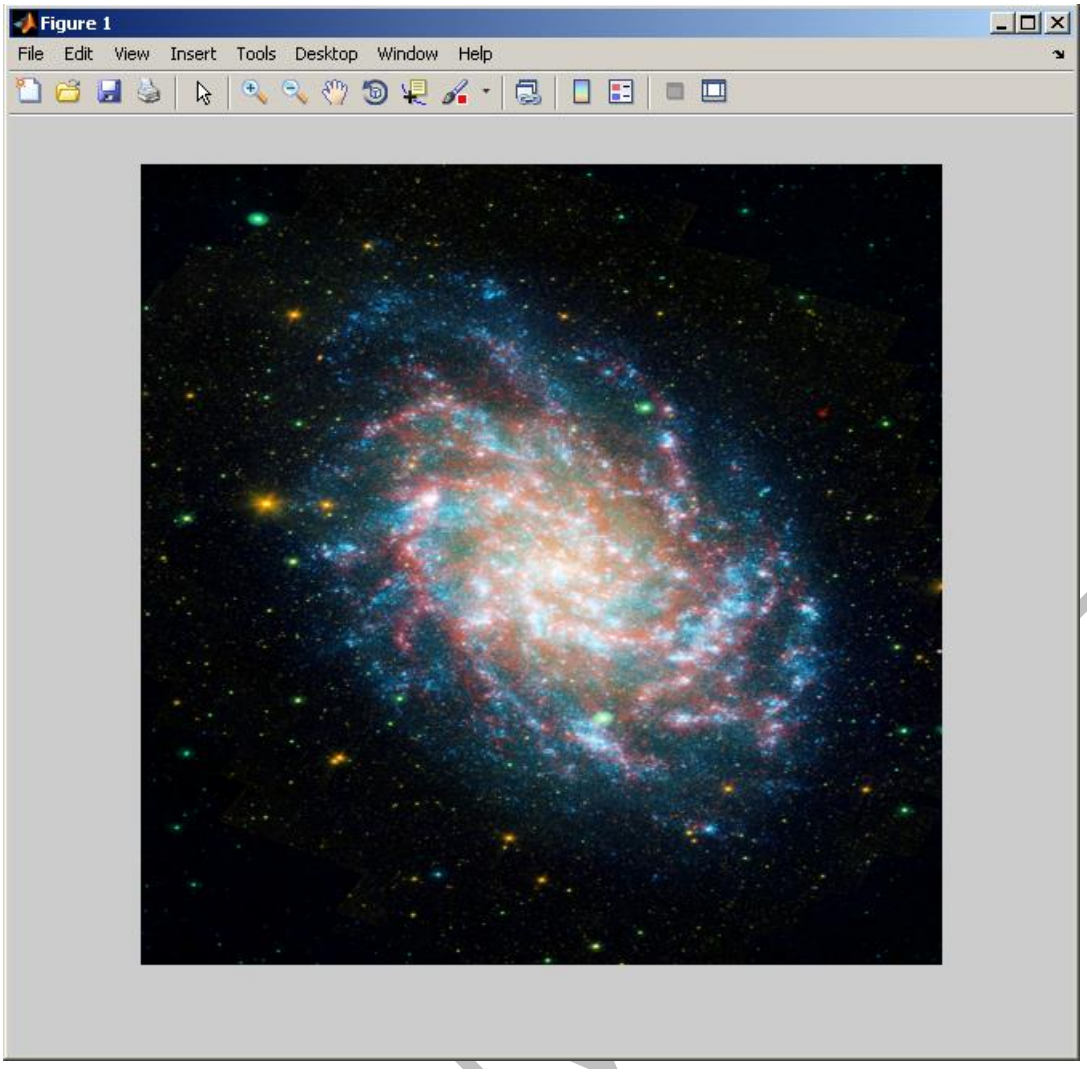
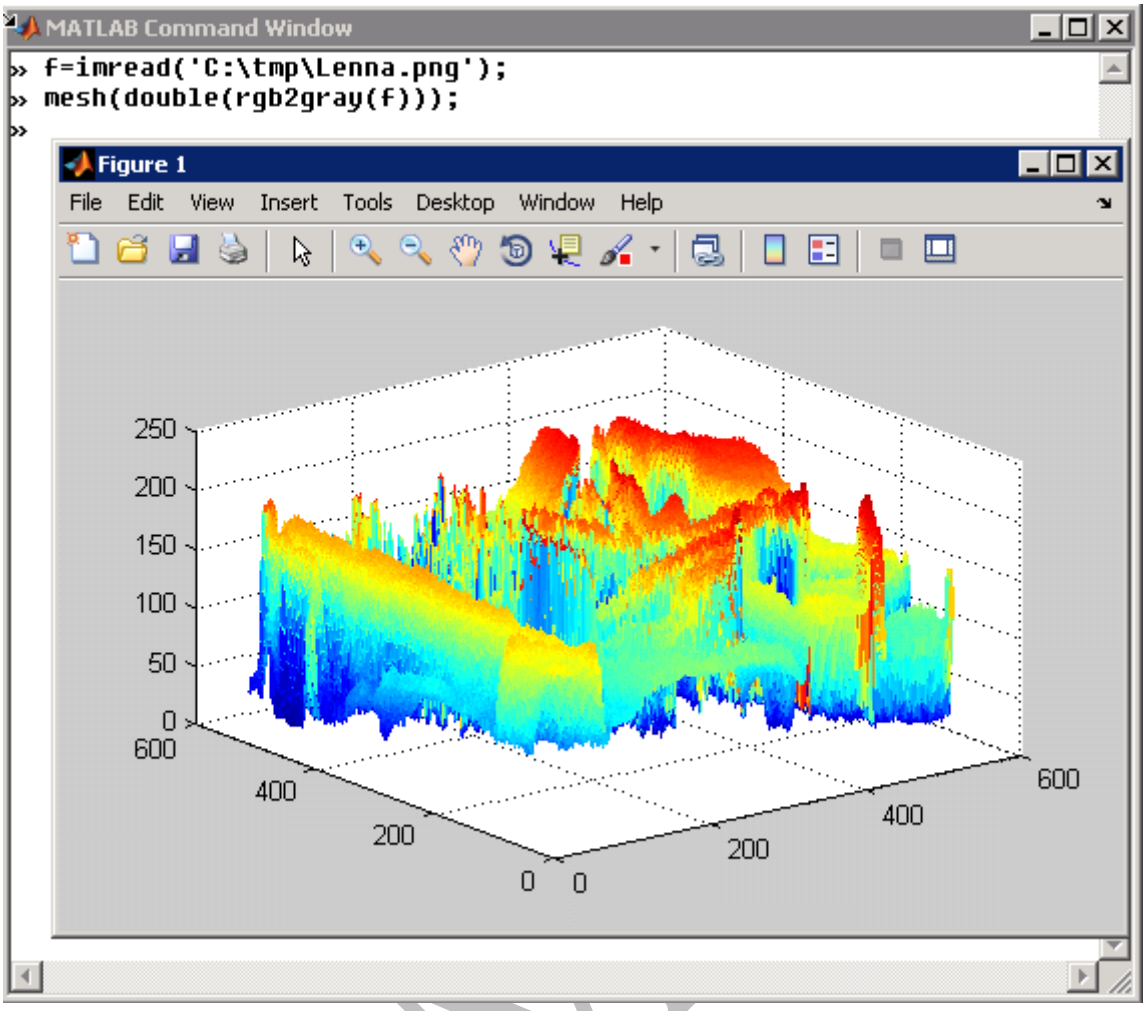


Image as a function

See the image as a function:

SAMPLE



If we change the perspective:

SAM

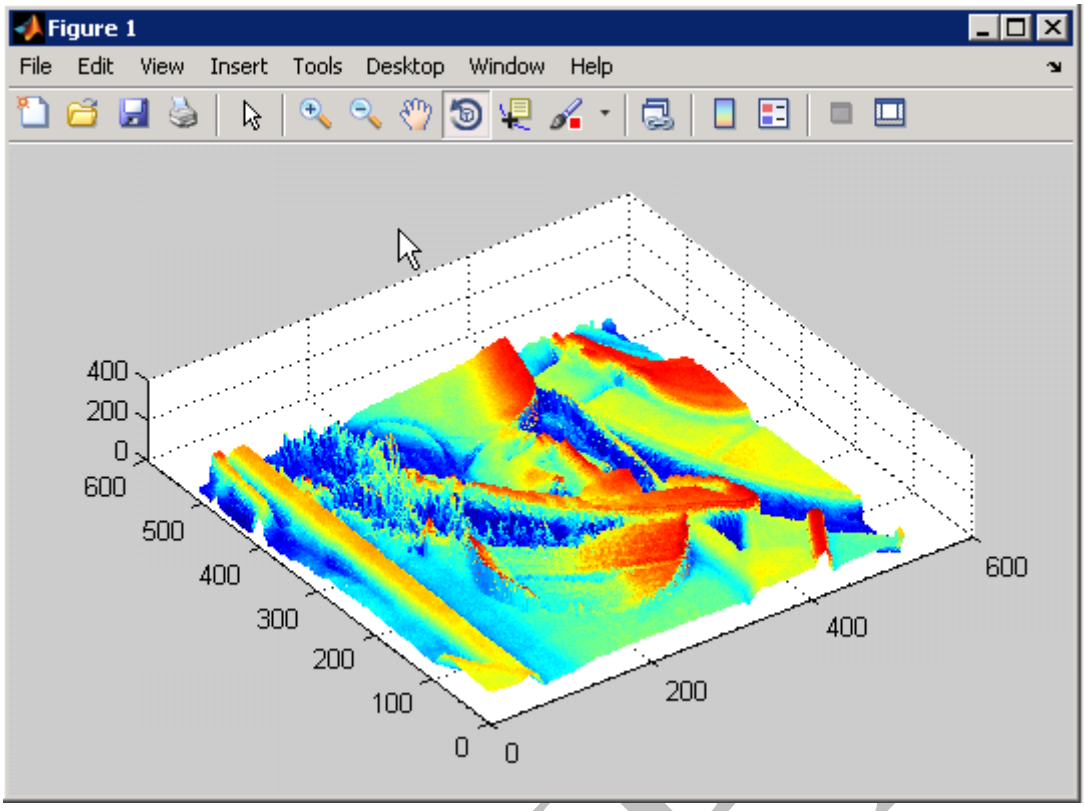
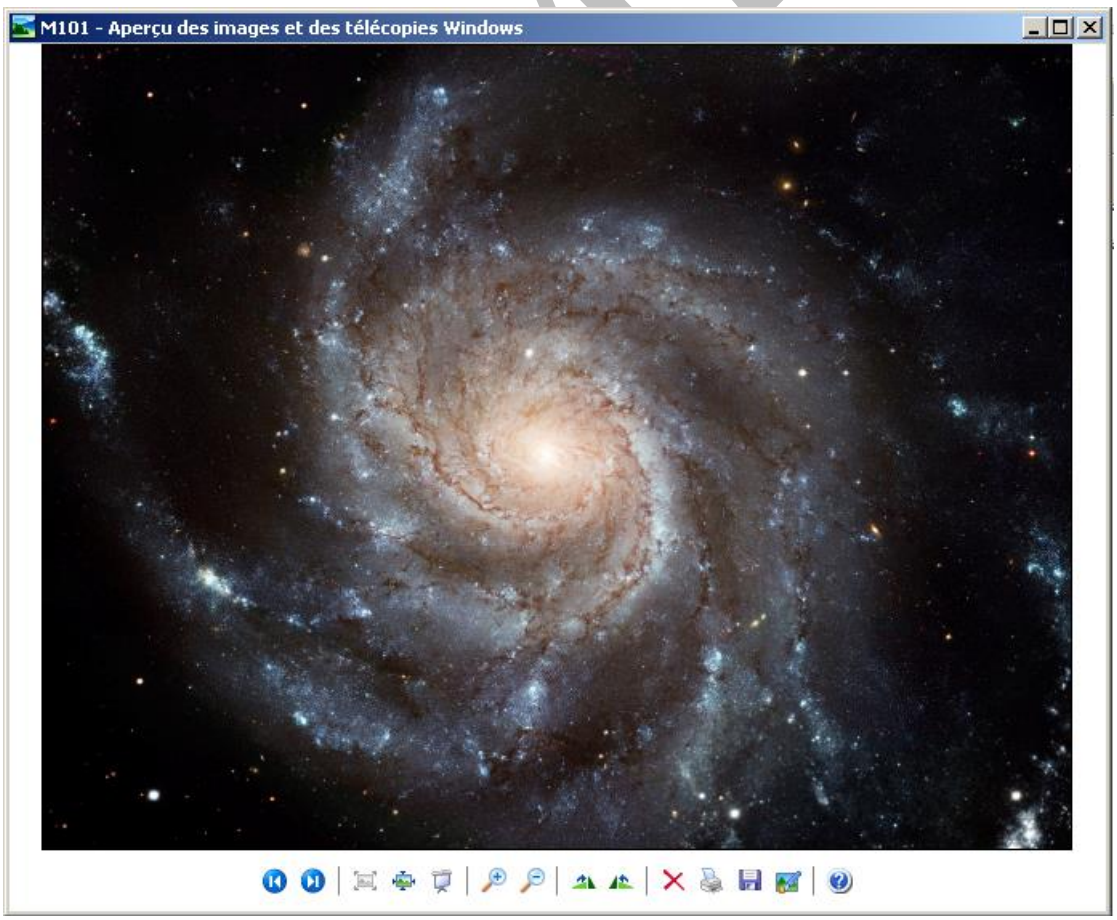


Image Isophobes



```
MATLAB Command Window
» img= imresize(imread('c:\tmp\M101.jpg'),[500, 500]);
» imagesc(img)
» hold on
» contour3(double(rgb2gray(img)),4)
» view(0,90)
»
```

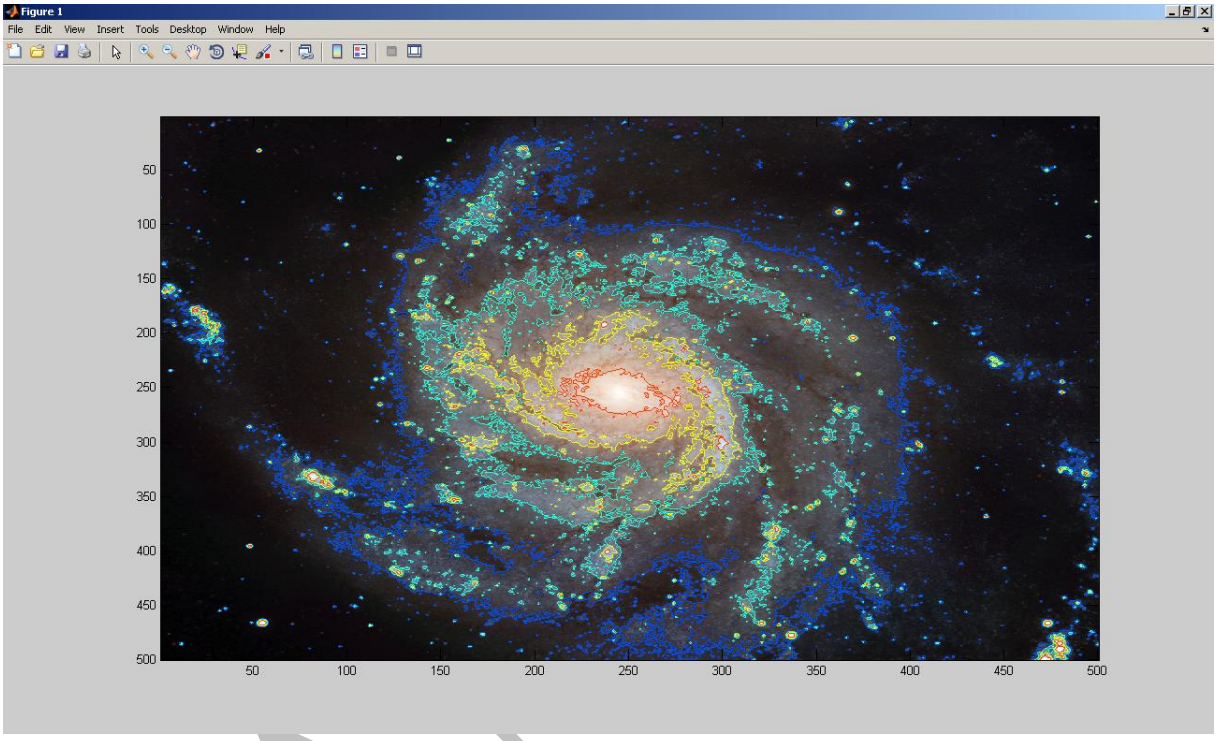
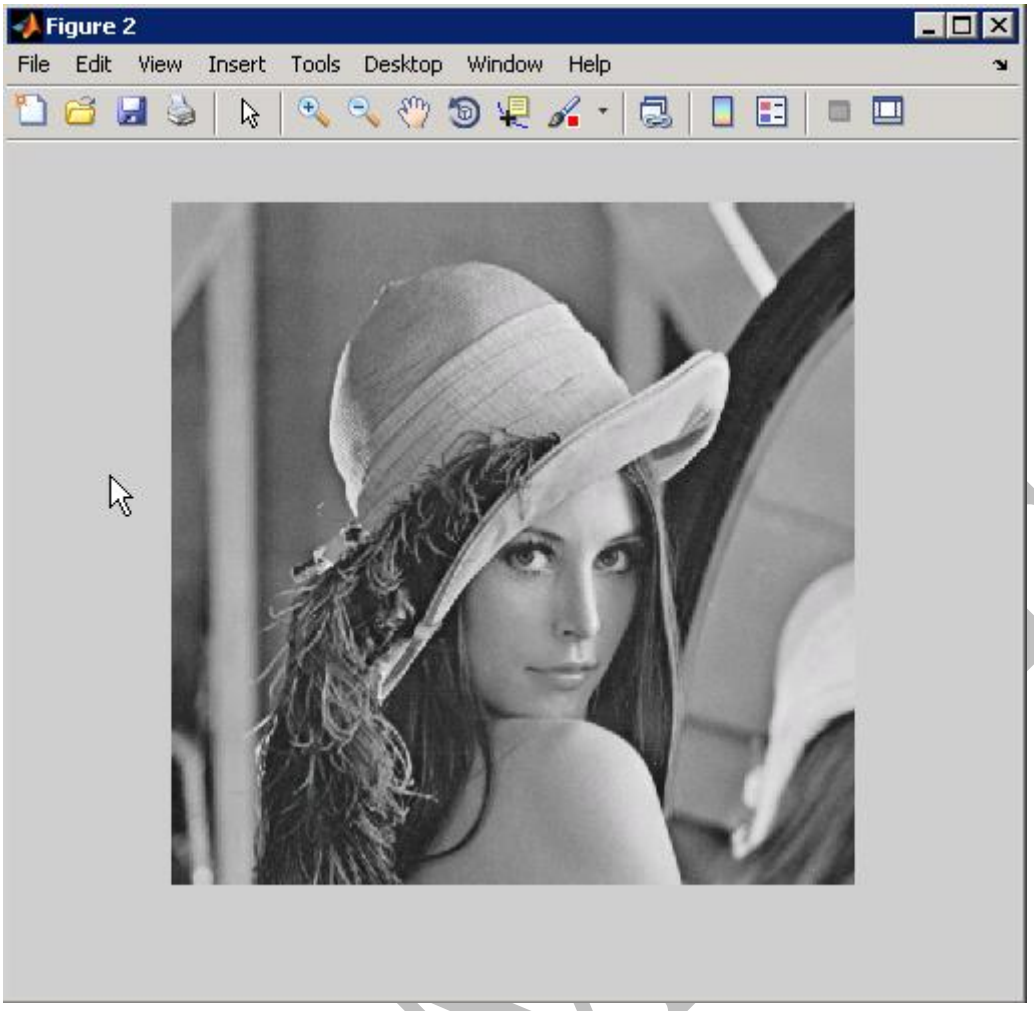
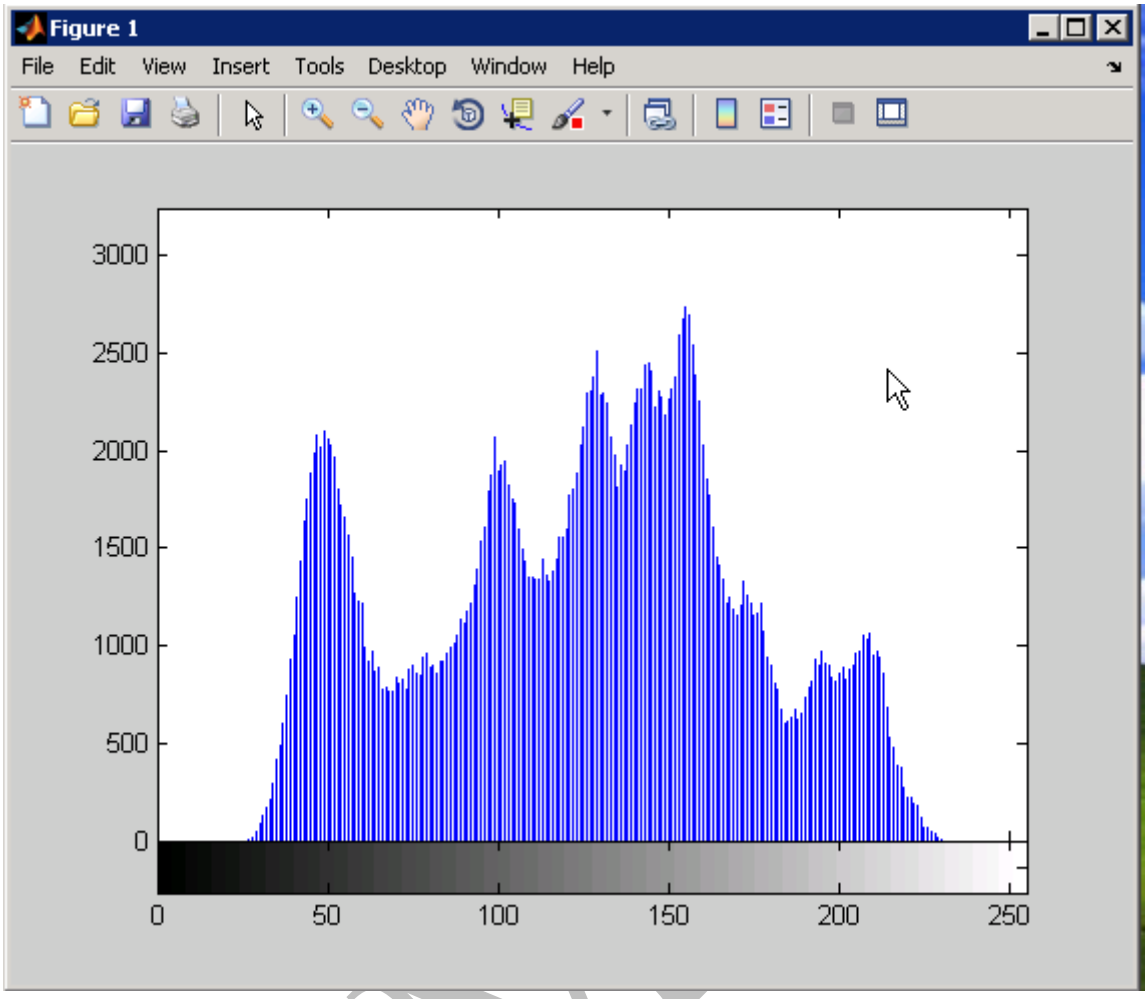


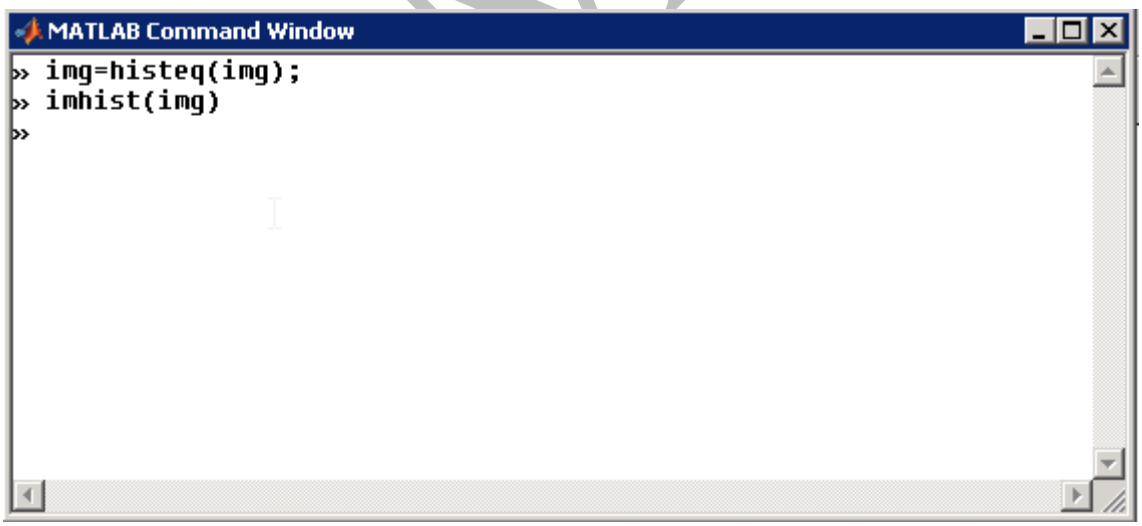
Image histogram

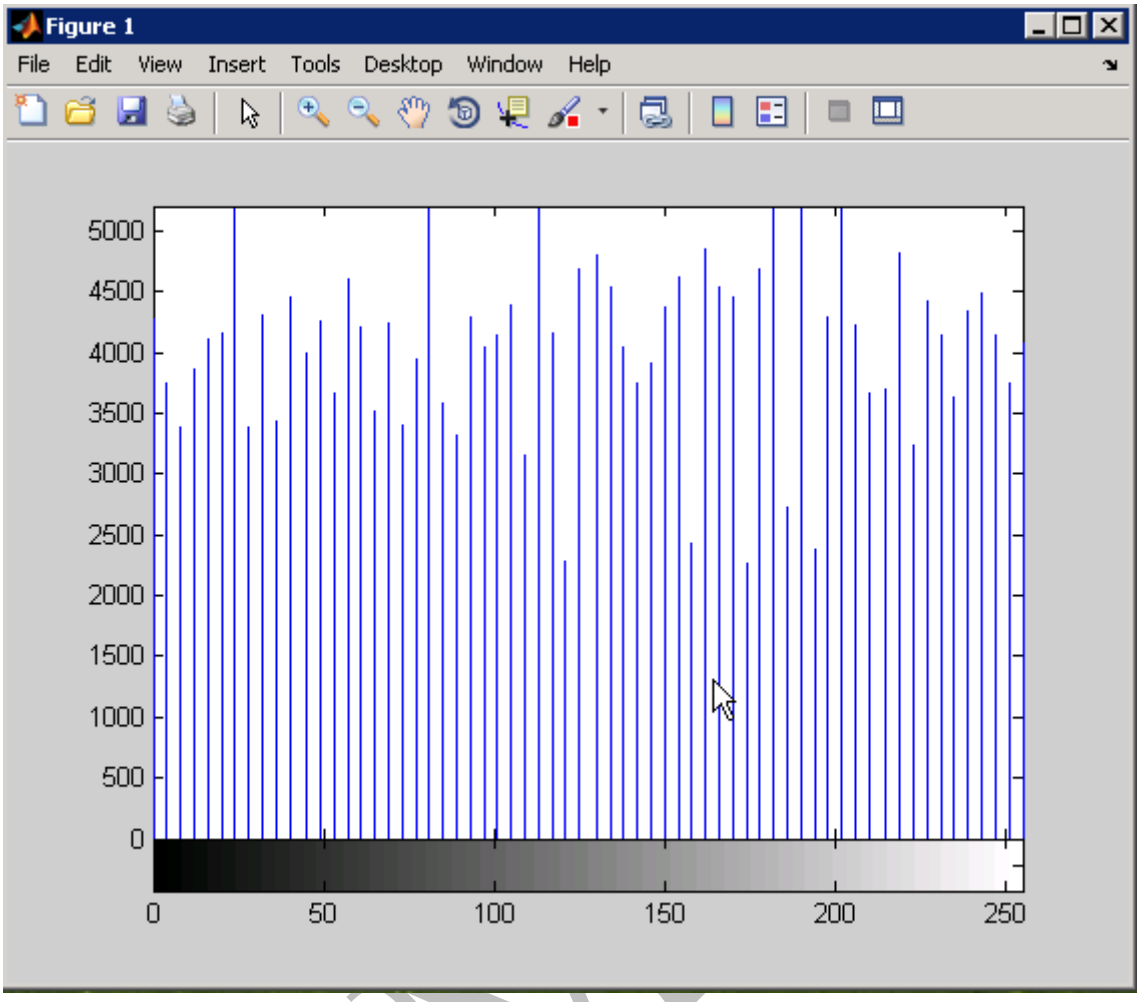
```
MATLAB Command Window
» img=imread('c:\tmp\Lenna.png');
» img=rgb2gray(img);
» imhist(img)
» figure
» imshow(img)
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 72
  In imshow at 283
»
```



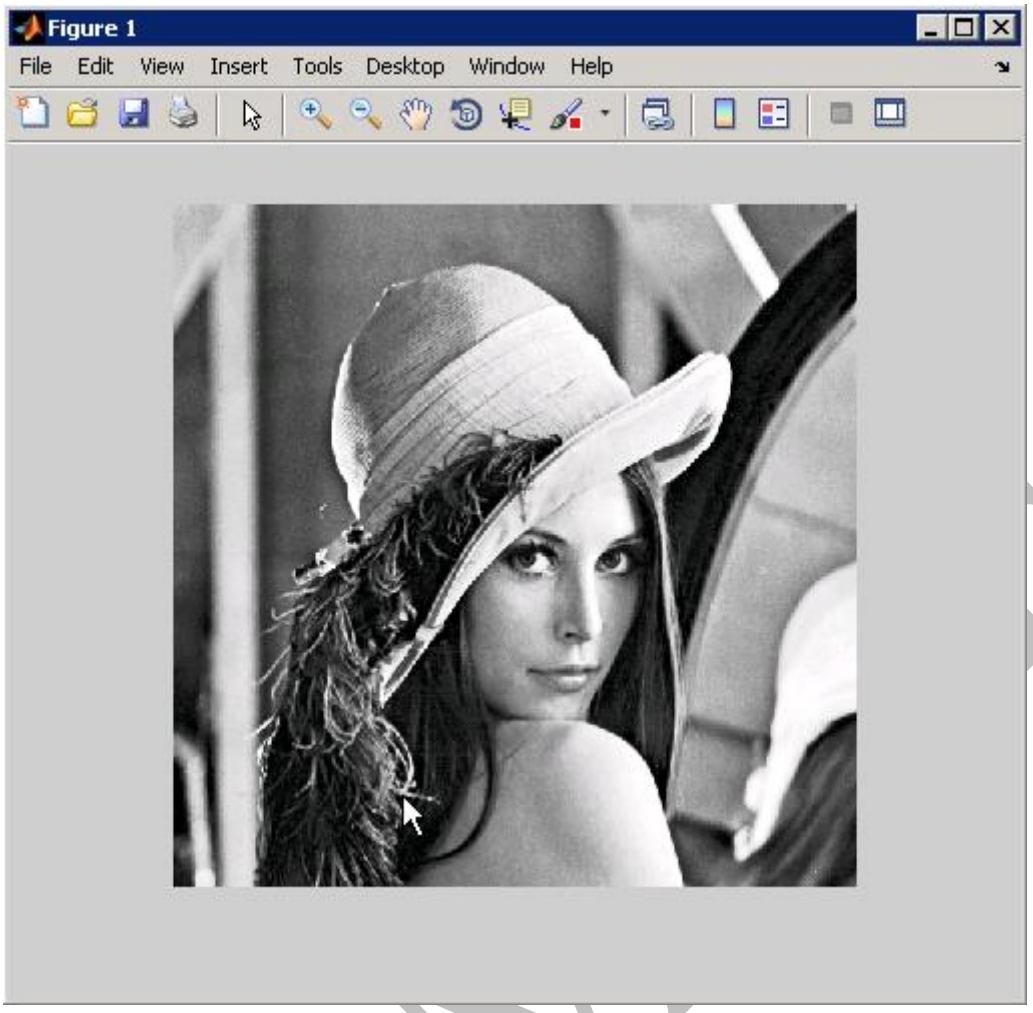
SAMPLE


Figure 1
File Edit View Insert Tools Desktop Window Help
3000
2500
2000
1500
1000
500
0
0 50 100 150 200 250


MATLAB Command Window
» img=histeq(img);
» imhist(img)
»

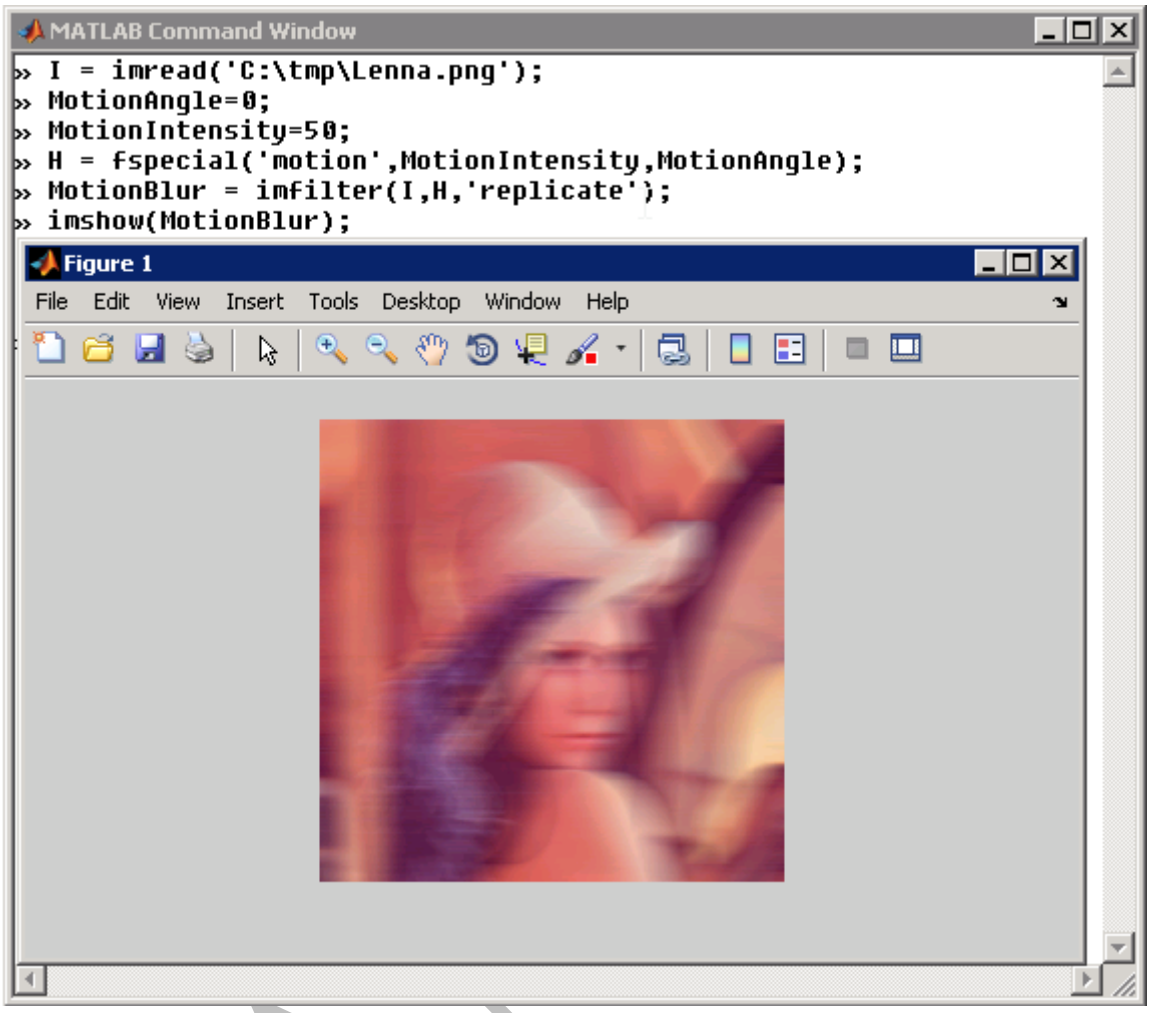

Figure 1
File Edit View Insert Tools Desktop Window Help
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0 50 100 150 200 250

SAM

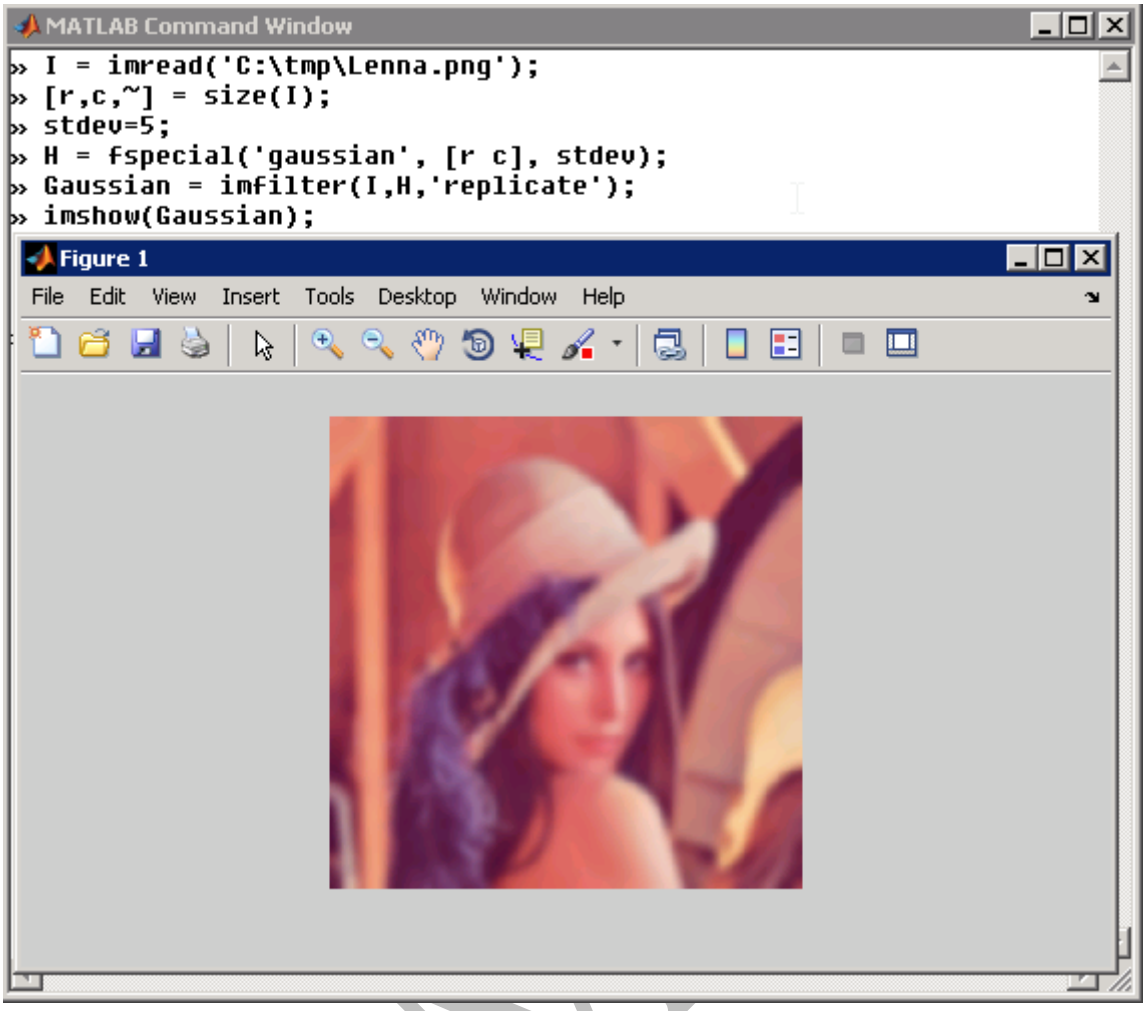


SAMPLE

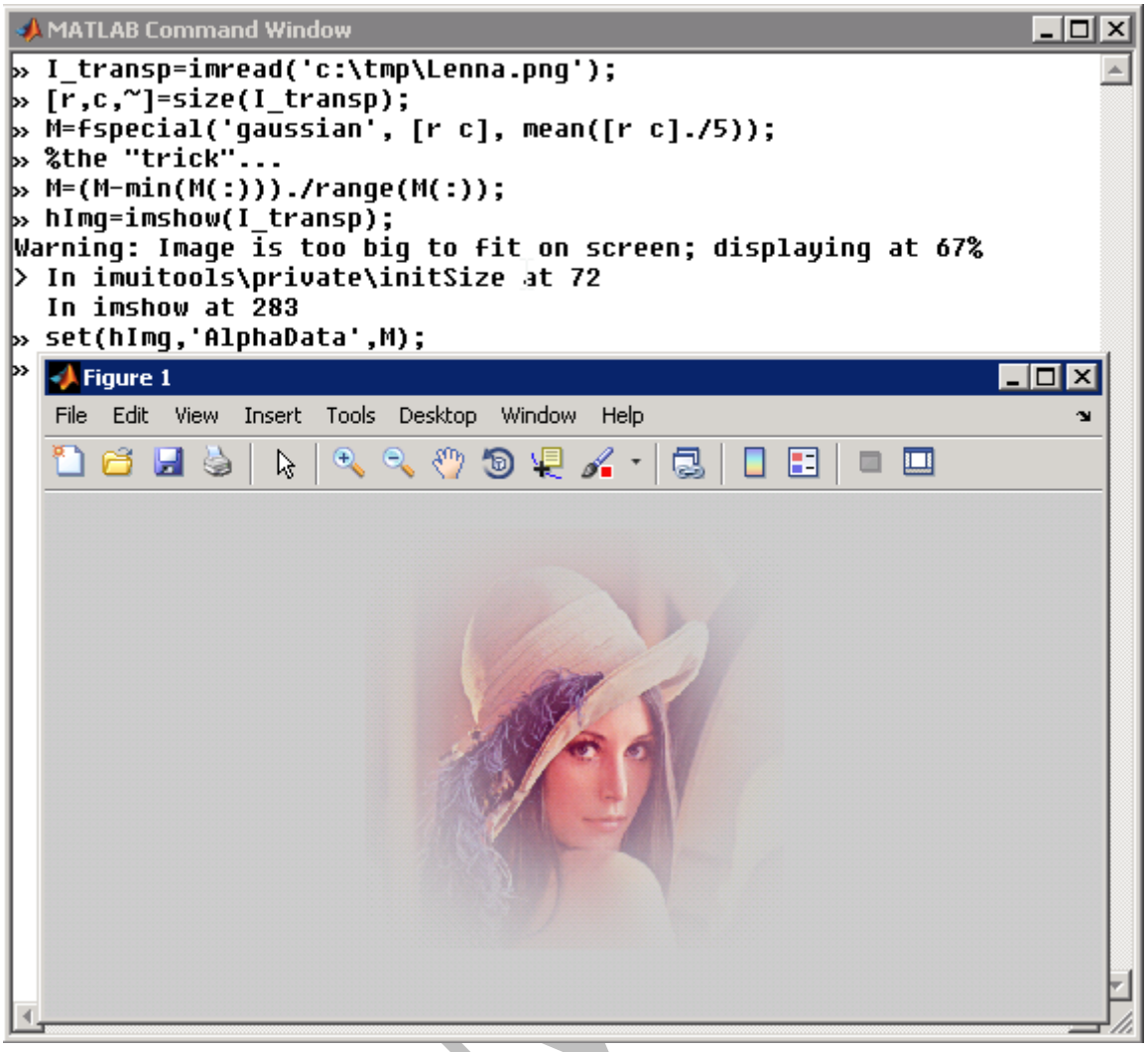
Filters (blur, motion) on an image



SAM


MATLAB Command Window
» I = imread('C:\tmp\Lenna.png');
» [r,c,~] = size(I);
» stdev=5;
» H = fspecial('gaussian', [r c], stdev);
» Gaussian = imfilter(I,H,'replicate');
» imshow(Gaussian);
Figure 1
File Edit View Insert Tools Desktop Window Help

SAM



Partially blur an image

Everybody knows what is the purpose of such applications. So let we see an example! This example comes from the "Two amigos" Mathworks contest.

Put a filter on an image can seem easy but actually with MATLAB it is not. See us how to do this:

```
I=imread('C:\tmp\Lenna.jpg');

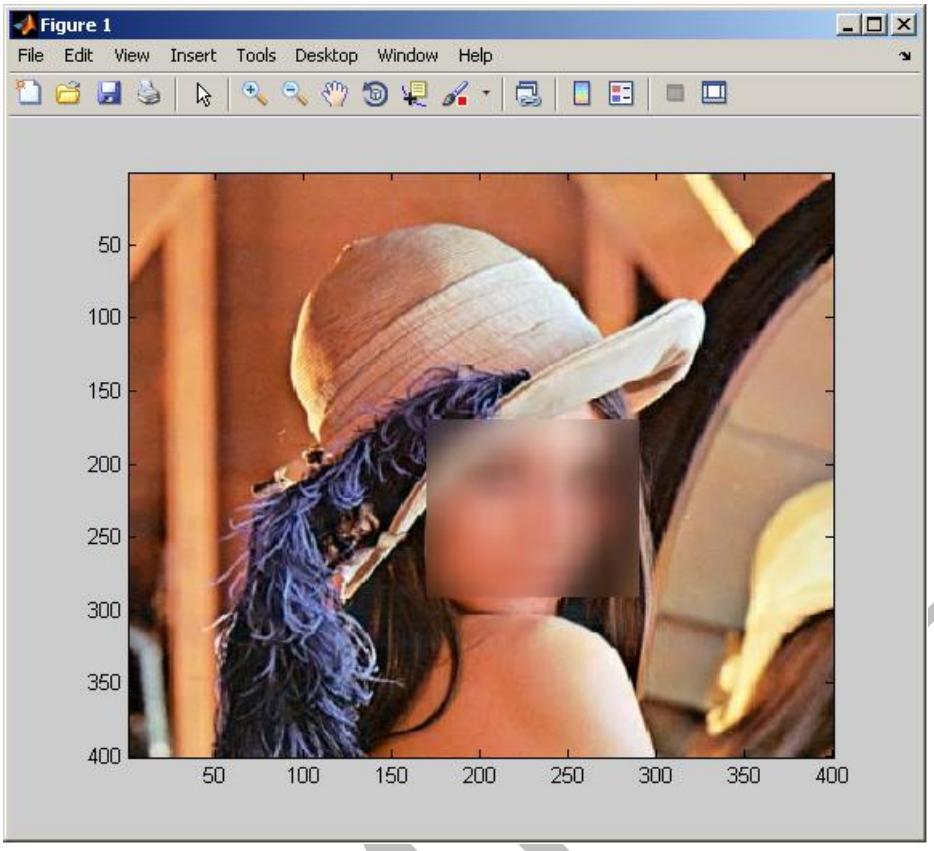
blurRadius = 20;
% Select 'Bob' region
bobRectX = 170:290;
bobRectY = 170:290;

% Calculate weights for the blur filter
w = [(blurRadius + 1):-1:1 2:(blurRadius + 1)];
w2 = repmat(w, length(w), 1);
weights = 1 ./ (w2 + w2');
weights = weights / sum(weights(:));

% Temporarily extend image bottom
bottom = repmat(I(end, :, : ), blurRadius, 1);
I = [I; bottom];
% Extend rectangle
extendedX = (min(bobRectX) - blurRadius):(max(bobRectX) + blurRadius);
extendedY = (min(bobRectY) - blurRadius):(max(bobRectY) + blurRadius);
% Apply filter to valid region
for i = 1:3
    I(bobRectX, bobRectY, i) = ...
       filter2(weights, I(extendedX, extendedY, i), 'valid');
end
% Remove the extension to the bottom of the image
I = I(1:(end - blurRadius), :, : );
image(I)
```

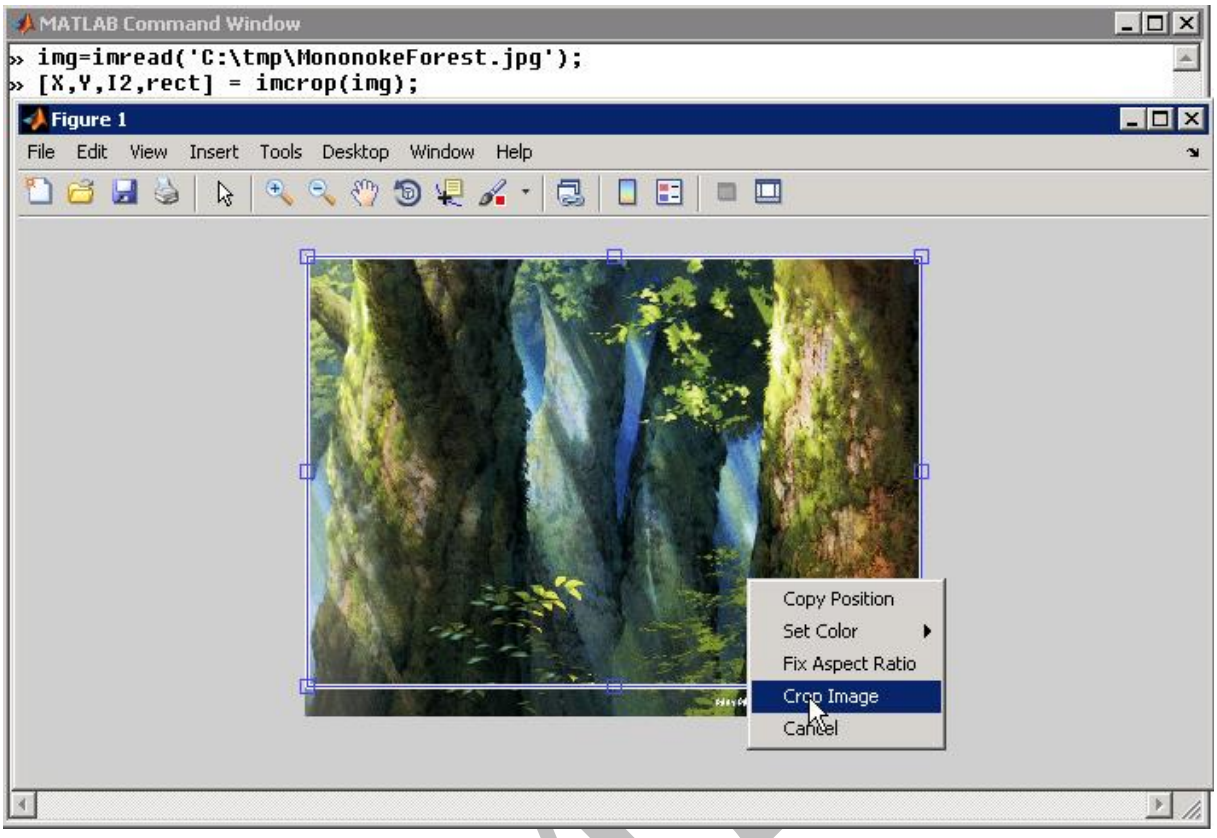
This gives:

SAM


Figure 1
File Edit View Insert Tools Desktop Window Help
50
100
150
200
250
300
350
400
50 100 150 200 250 300 350 400

SAMPLE

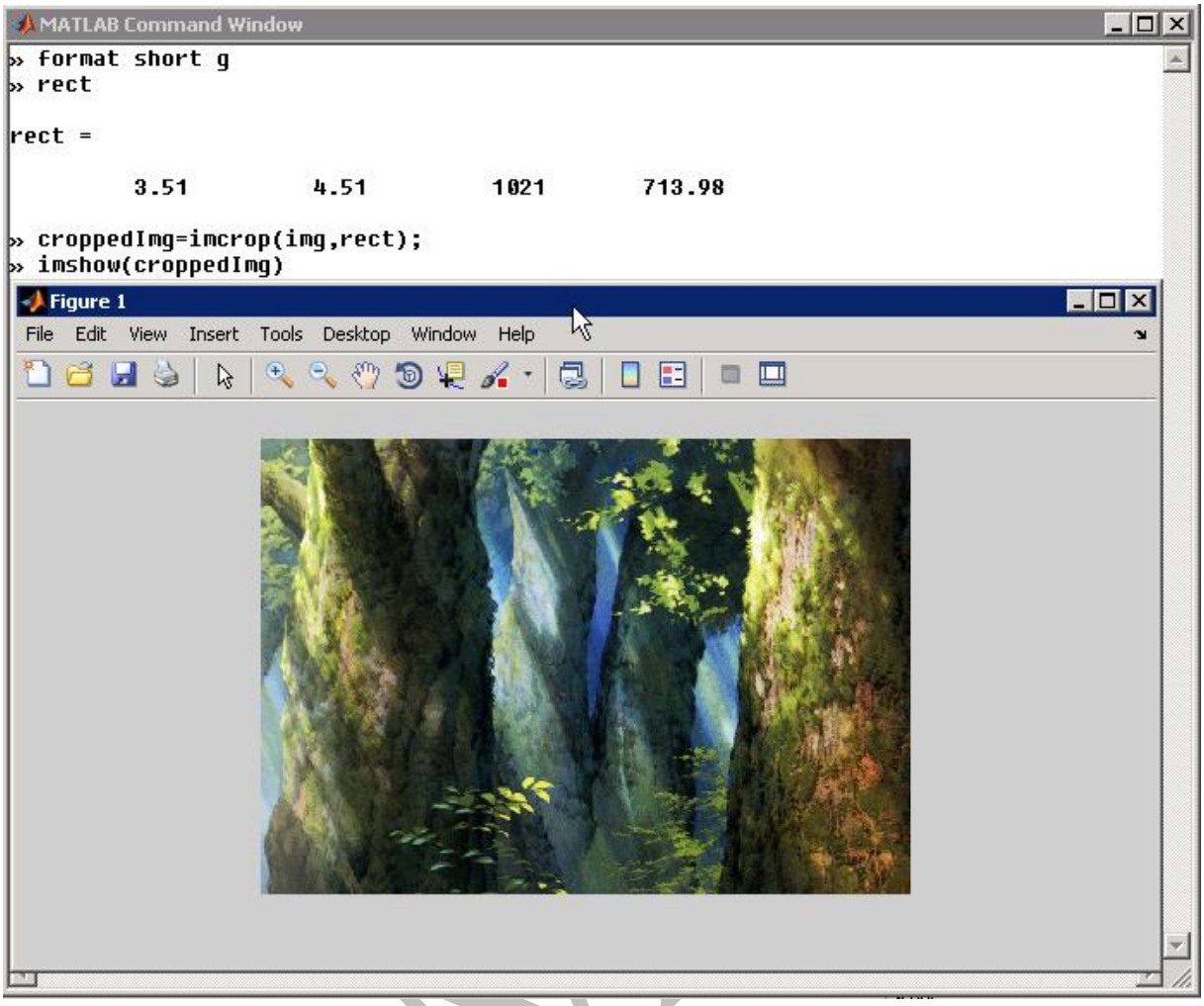
Crop an image



We can also crop the image be specifying the area:

imcrop(img,[xmin ymin width height])

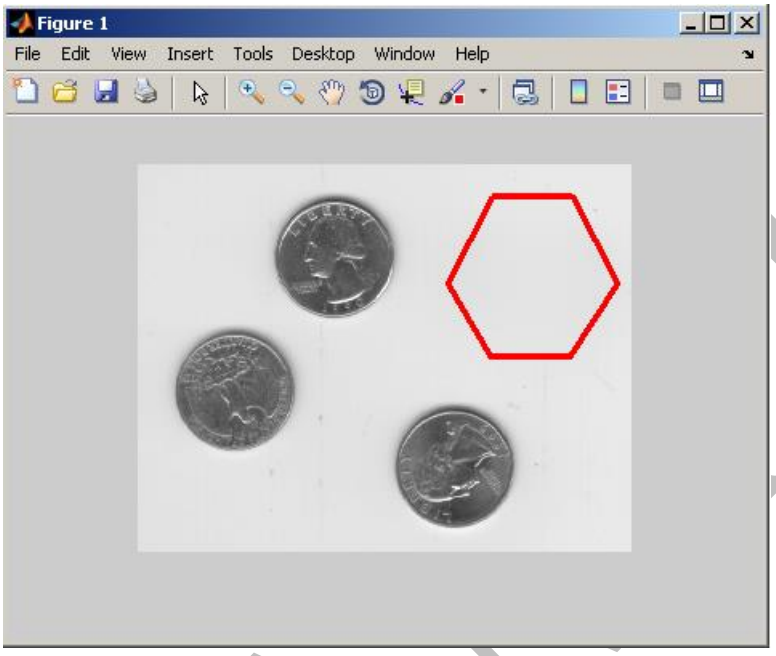
SAMPLE



Region of interest



```
%Fill in specified region of interest (ROI) polygon in grayscale image
clear all;clc
I = imread('eight.tif');
c = [222 272 300 270 221 194 222];
r = [21 21 75 121 121 75 21];
J = roifill(I,c,r);
figure, imshow(J)
hold on;
plot(c,r,'LineWidth',3,'Color','r');
```

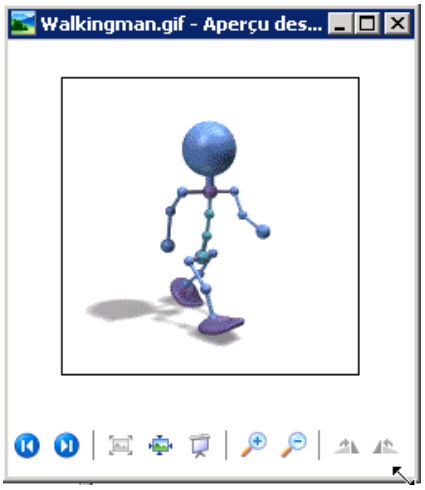


*Convert a *.gif file into an *.avi*

This is sometimes a request of some MATLAB users that have some animated chart as *.gif without having the original source code or data and have then no other choices than convert the *.gif into *.avi to have video.

Suppose we have the following animated *.gif file for the example:

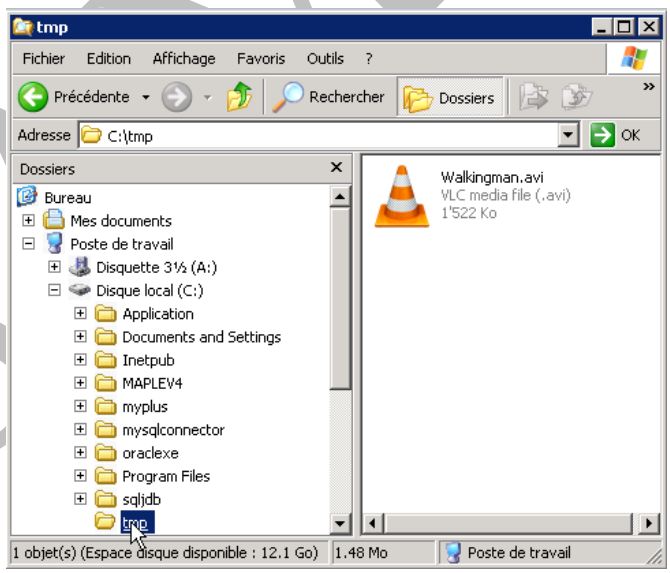
SAMPLE



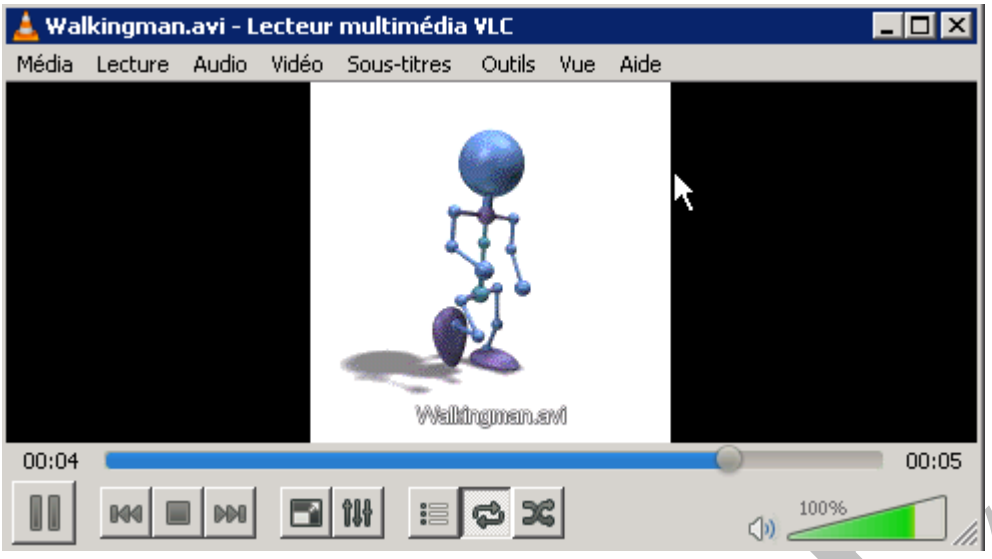
The script will use the function **immovie( )**:

```
%convert a *.gif to a move file without compression and 3 frames per
%second
[A,map]=imread('E:\Walkingman.gif','frames','all');
mov=immovie(A,map);
movie2avi(mov,'C:\tmp\Walkingman.avi','compression', 'None','fps',3);
```

Then we get a new file:



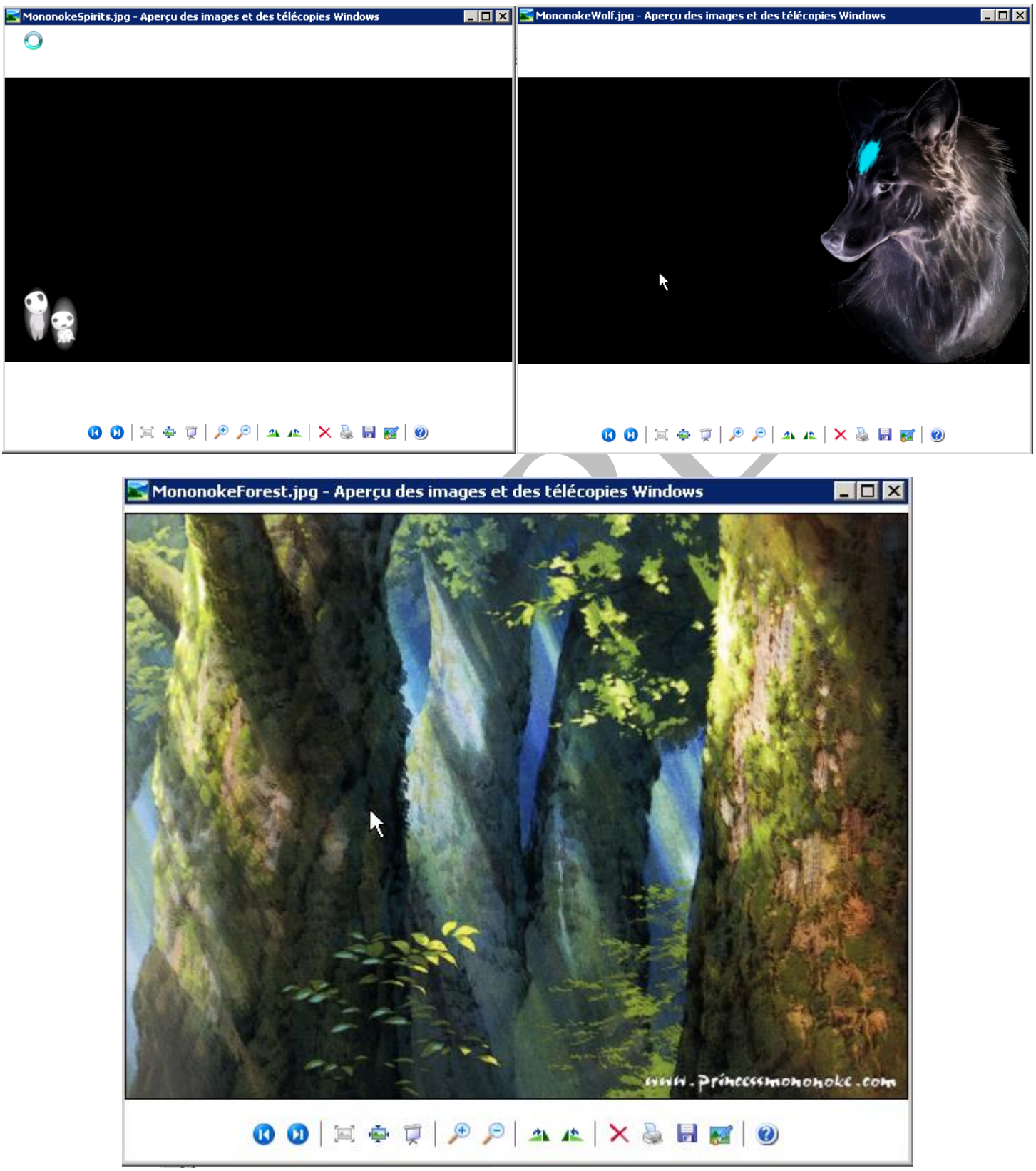
And if we play it:



Job done!

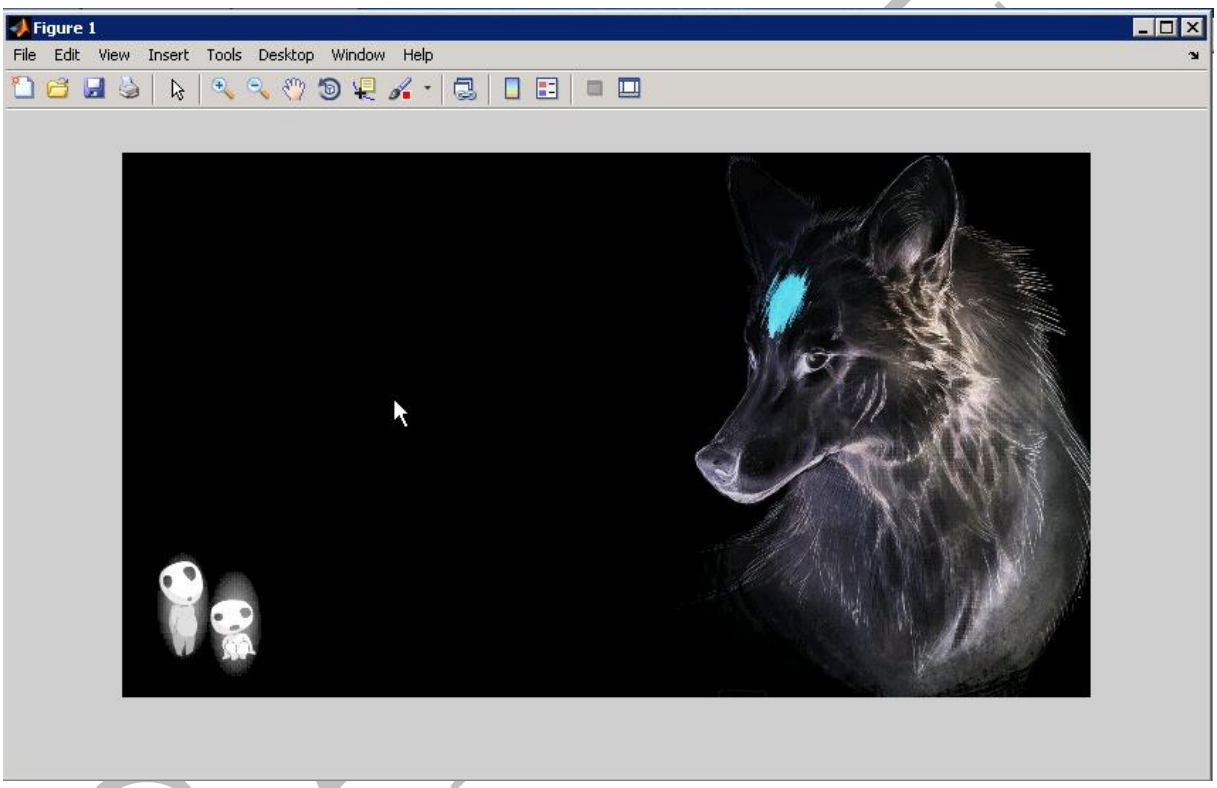
SAMPLE

Overlay images with masks



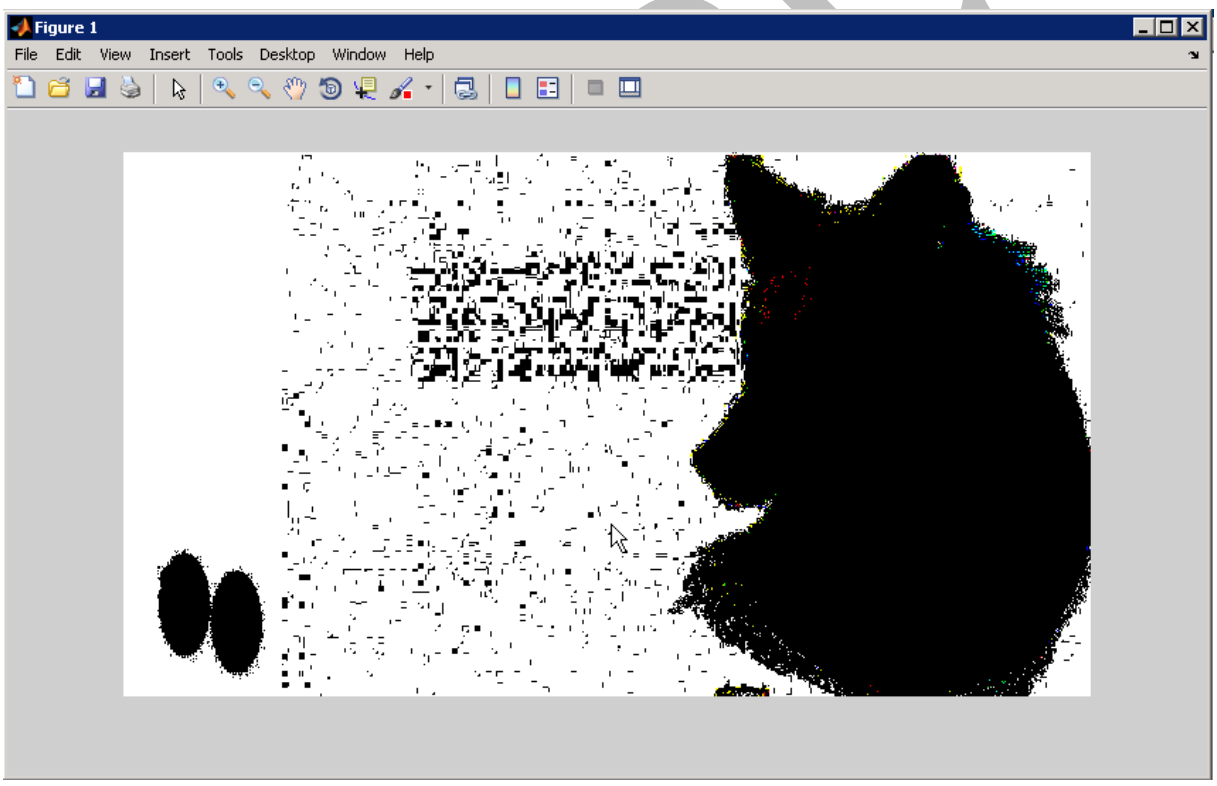
EDITOR PUBLISH VIEW

```
1 -    background=imread('C:\tmp\MononokeForest.jpg');
2 -    imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
3 -    imgWolf=imread('C:\tmp\MononokeWolf.jpg');
4
5      %Image addition
6      %Both A and B are of same size
7 -    imgOver=imgSpirits+imgWolf;
8 -    imshow(imgOver)
```



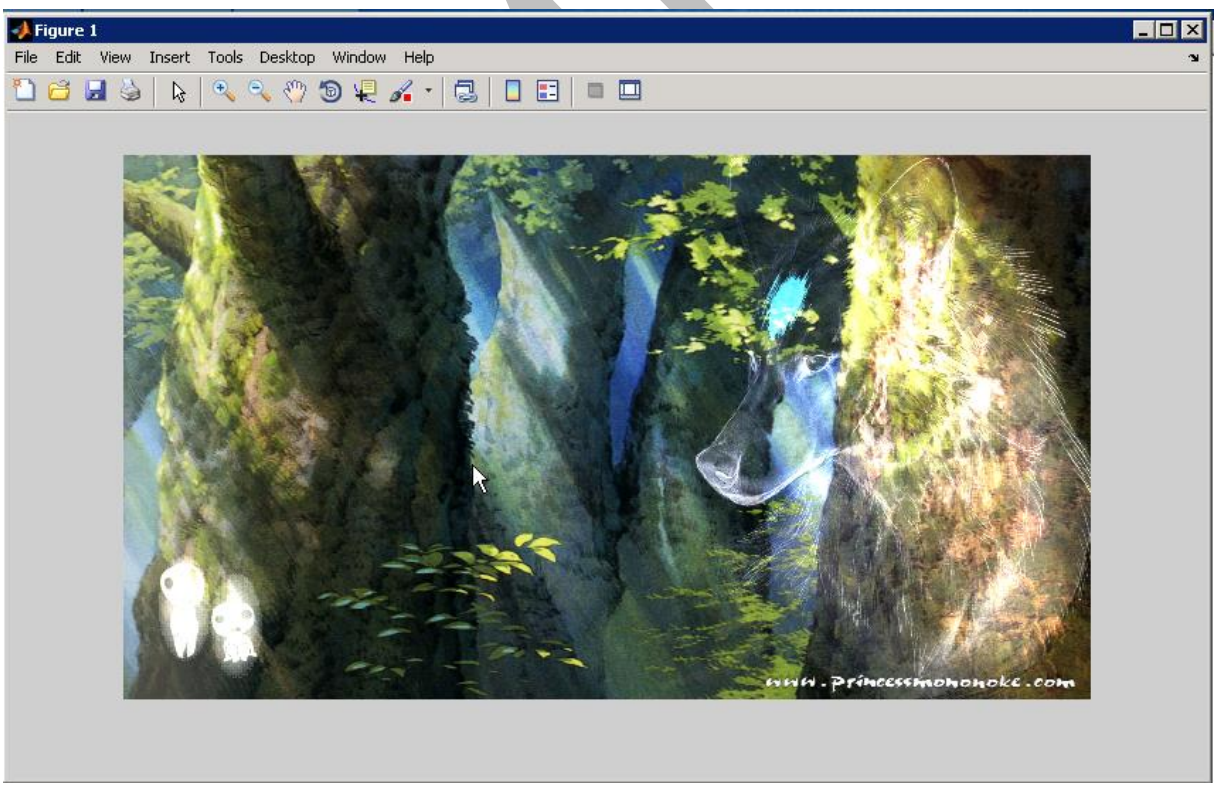
EDITOR PUBLISH VIEW

```
1 -    close all;clear all;clc
2 -    background=imread('C:\tmp\MononokeForest.jpg');
3 -    imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
4 -    imgWolf=imread('C:\tmp\MononokeWolf.jpg');
5
6      %Image addition
7      %Both A and B are of same size
8 -    imgOver=imgSpirits+imgWolf;
9
10     %we create the mask
11 -    background=imresize(background,[size(imgOver,1) size(imgOver,2)]);
12 -    Im3=uint8(zeros(size(imgOver)));
13 -    whiteImg=uint8(ones(size(imgOver)));
14 -    mask=whiteImg./imgOver;
15 -    im3=uint8(mask&imgOver);%uint8(and(mask,background));
16 -    imshow(mask)
```

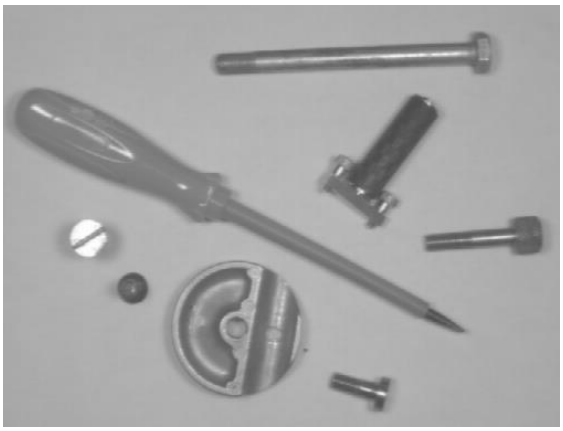


EDITOR PUBLISH VIEW

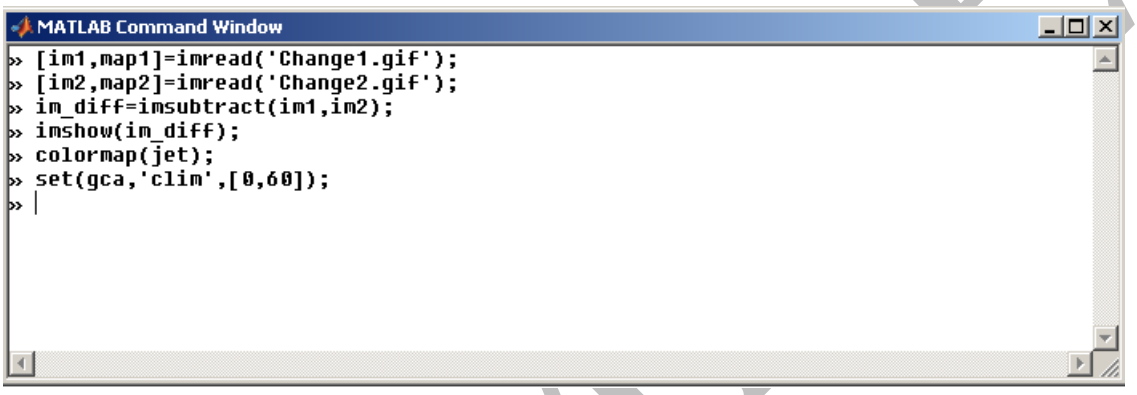
```
1  close all;clear all;clc
2  background=imread('C:\tmp\MononokeForest.jpg');
3  imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
4  imgWolf=imread('C:\tmp\MononokeWolf.jpg');
5
6  %Image addition
7  %Both A and B are of same size
8  imgOver=imgSpirits+imgWolf;
9
10 %we create the mask
11 background=imresize(background,[size(imgOver,1) size(imgOver,2)]);
12 Im3=uint8(zeros(size(imgOver)));
13 whiteImg=uint8(ones(size(imgOver)));
14 mask=whiteImg./imgOver;
15 im3=uint8(mask&imgOver);%uint8(and(mask,background));
16
17 %we add the two images
18 finalImg=(background)+imgOver;
19 imshow(finalImg);
```

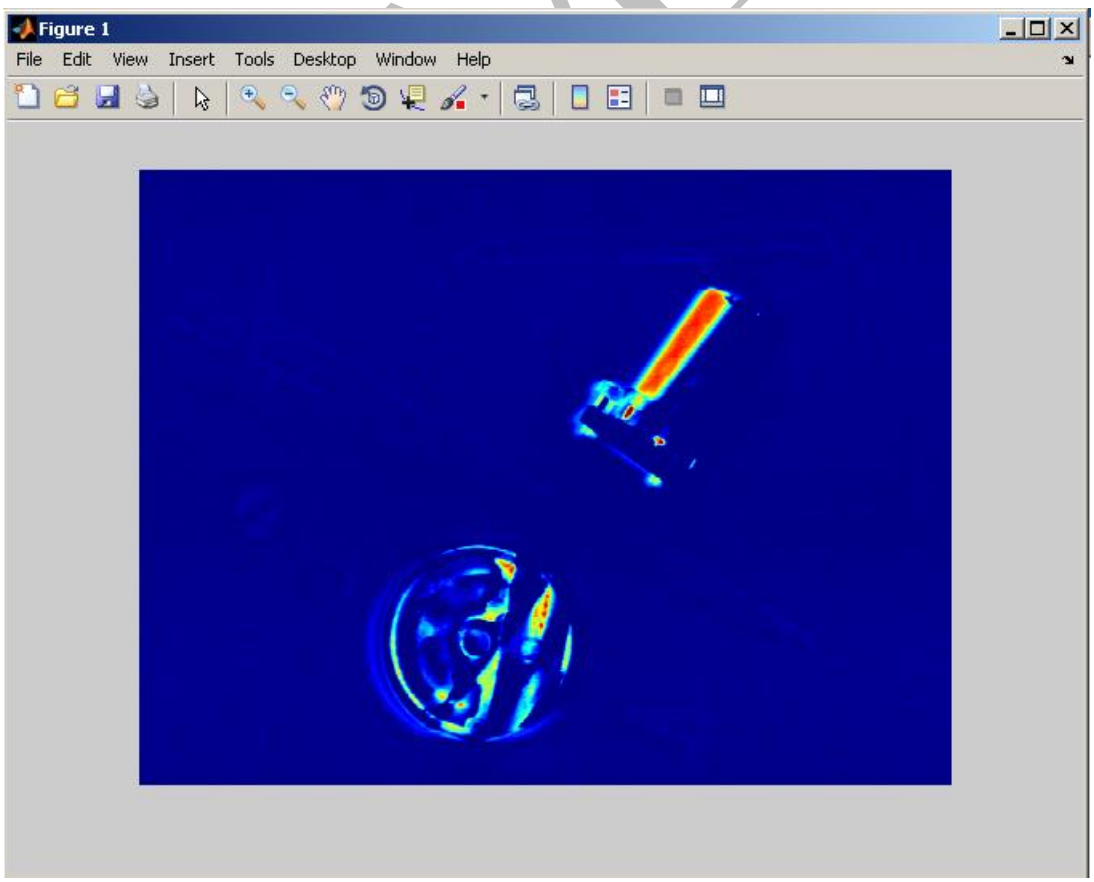


Compare Images in colormapping



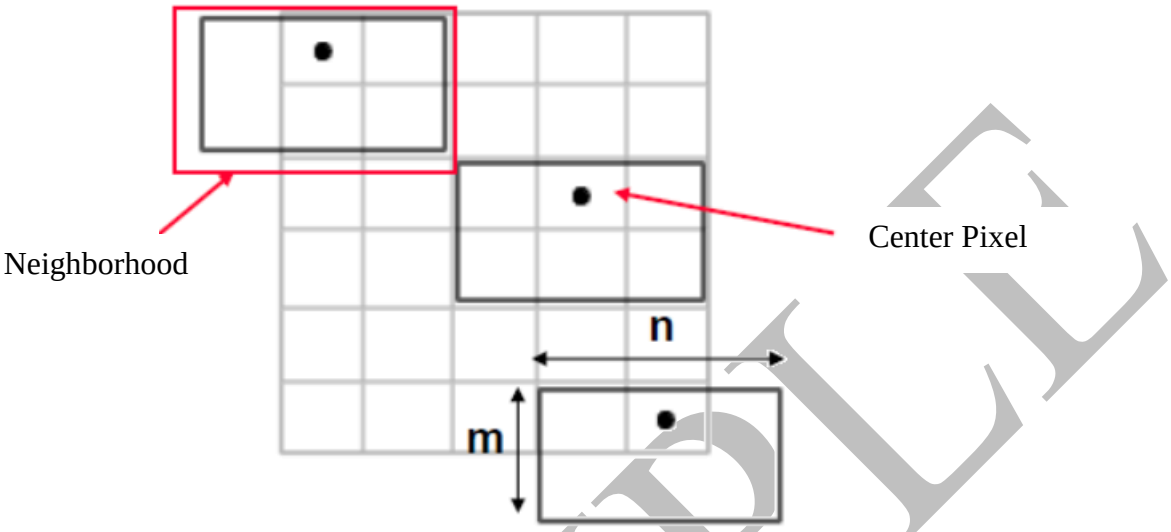




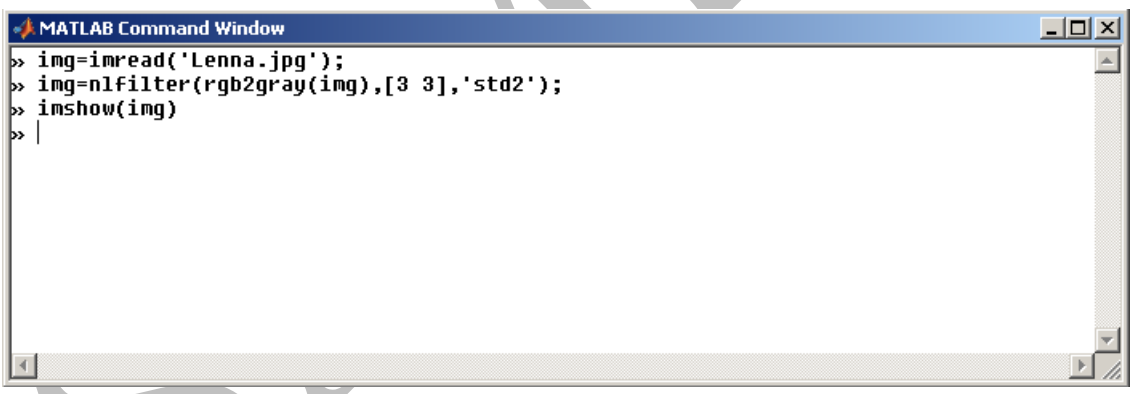


Sliding Neighborhood Operations

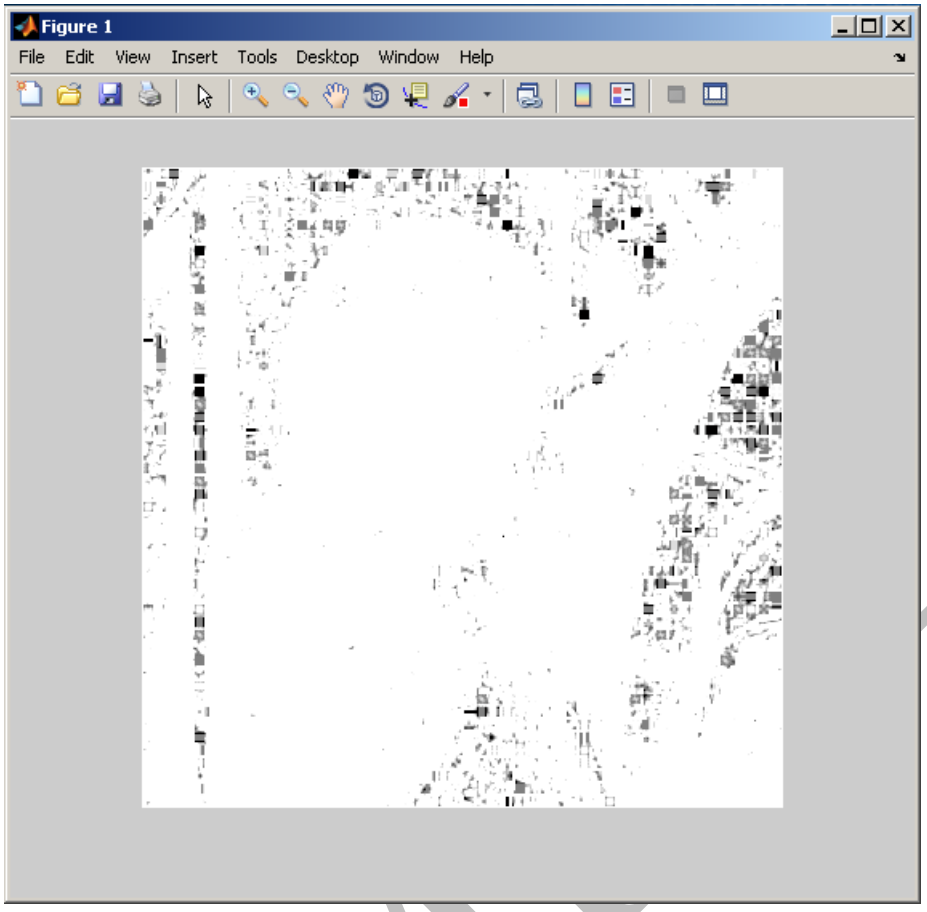
Each center pixel value is determined by applying some algorithm to its neighboring pixels of a defined size – neighborhood.



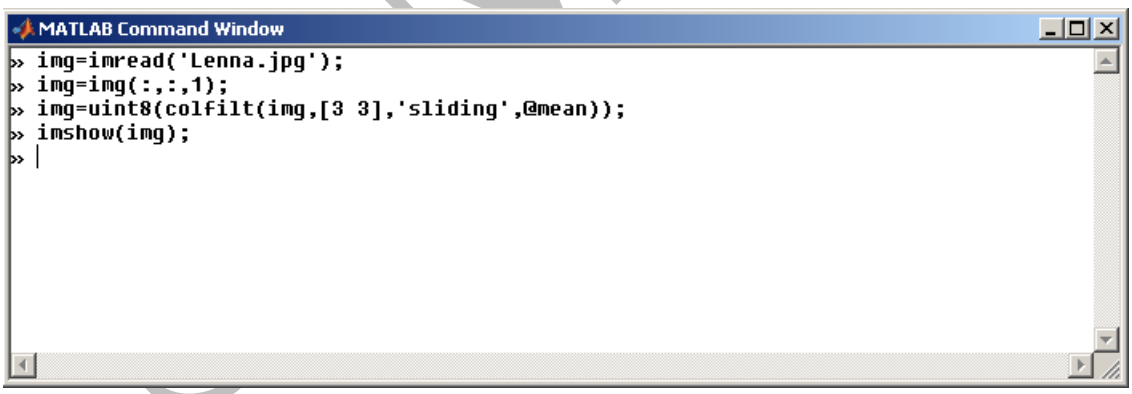
nlfilter() works only with on color layer at a time or with gray/color images:

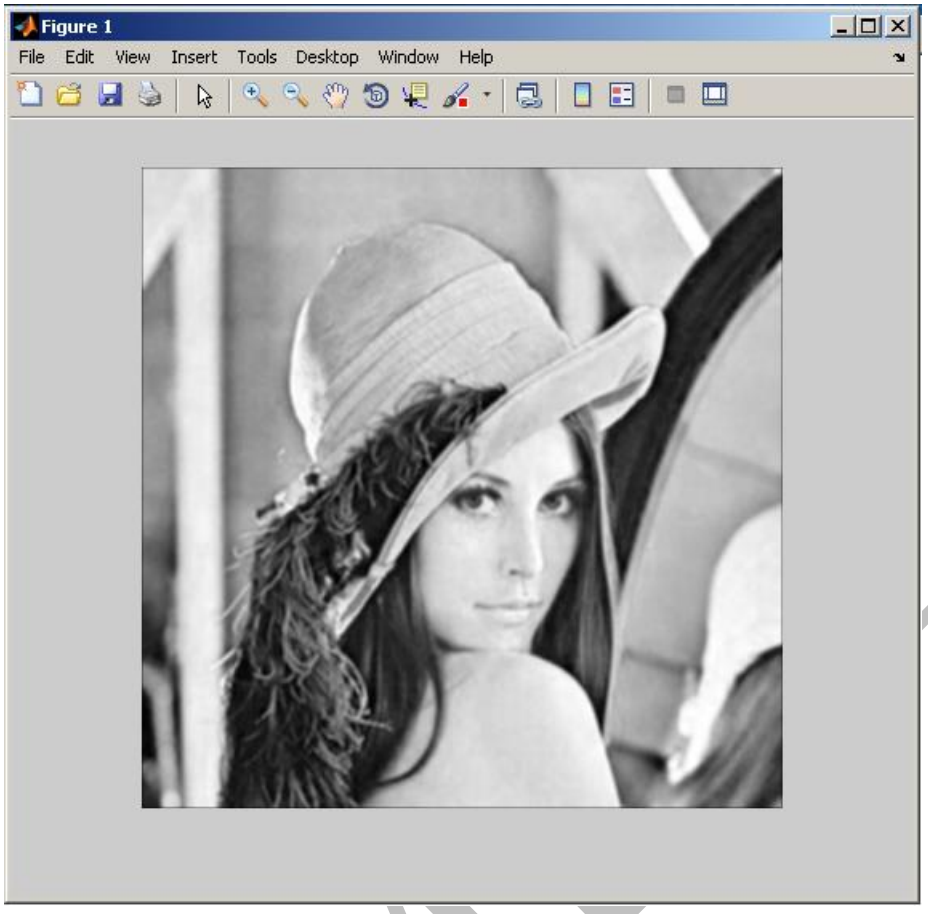


SAMPLE



Or with a much more faster function and other statistical filter by application on only one given color layer:

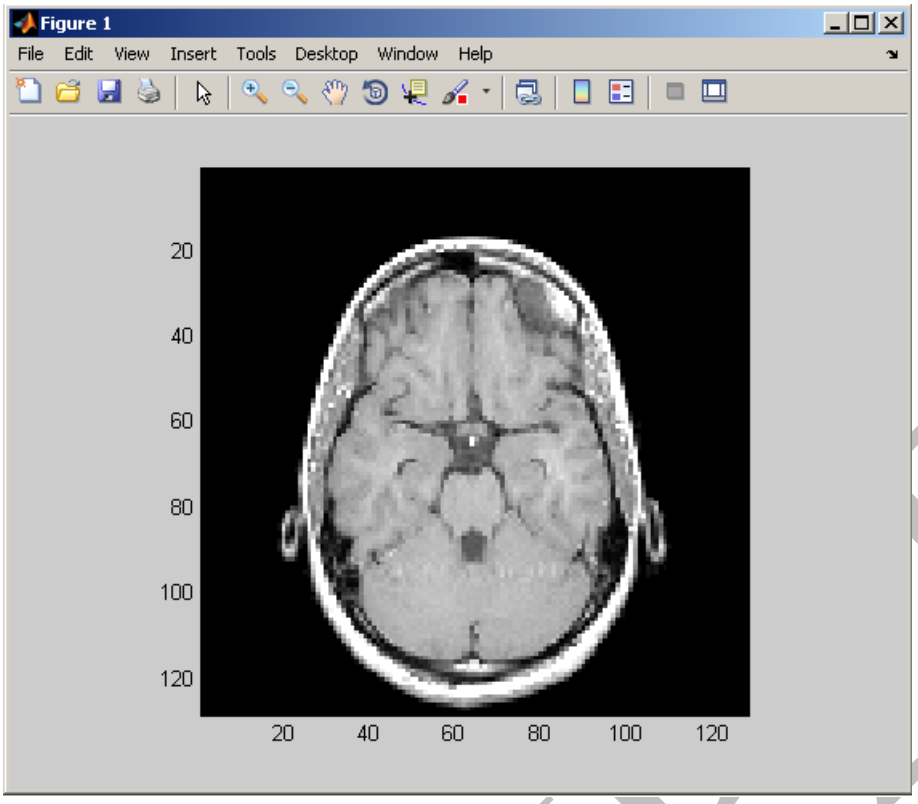




MRI (Magnetic Resonance Imaging) Dataset

EDITOR PUBLISH VIEW

```
1   %load MRI dataset: 27 slices of 128x128 images
2   load mri
3   D = squeeze(D); % 27 2D-images
4
5   %#view a given slice
6   figure('Colormap',map)
7   slice_number_to_view=9;
8   image(D(:,:,slice_number_to_view))
9   axis image
```



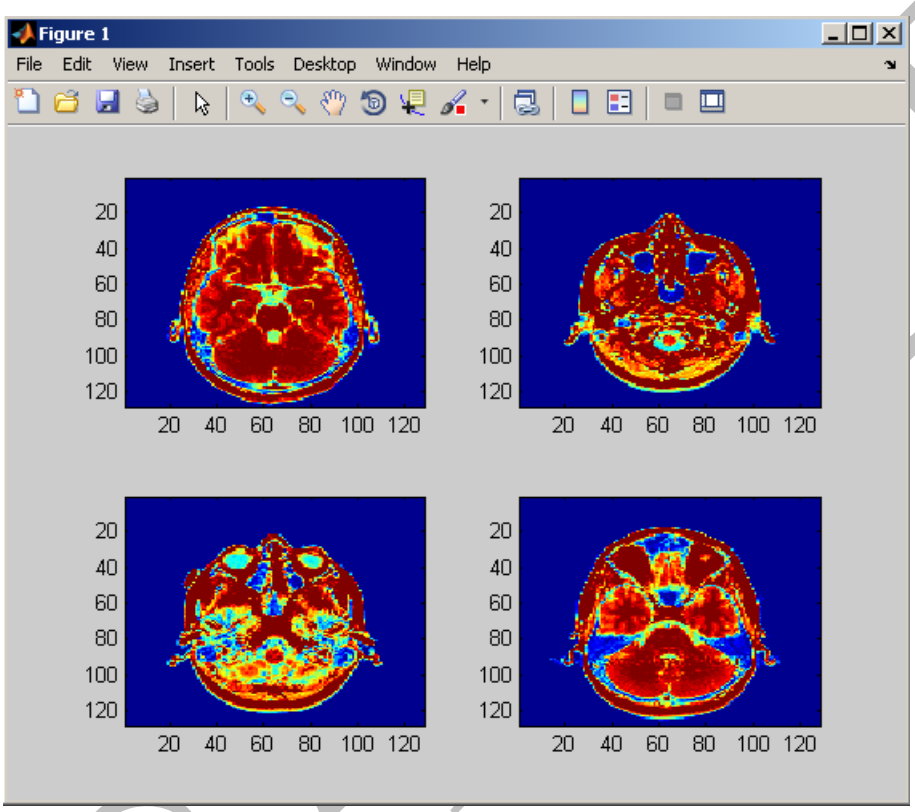
Animate all images on over the other:

```
EDITOR  PUBLISH  VIEW
1   load mri;
2   D=squeeze(D);
3   [N, M, k]=size(D);
4   for i=1:k;
5       figure,image(D(:,:,i));
6       axis image;
7       colormap(map);
8   end;
9   close all;
```

Or:

```
load mri;
D=squeeze(D);

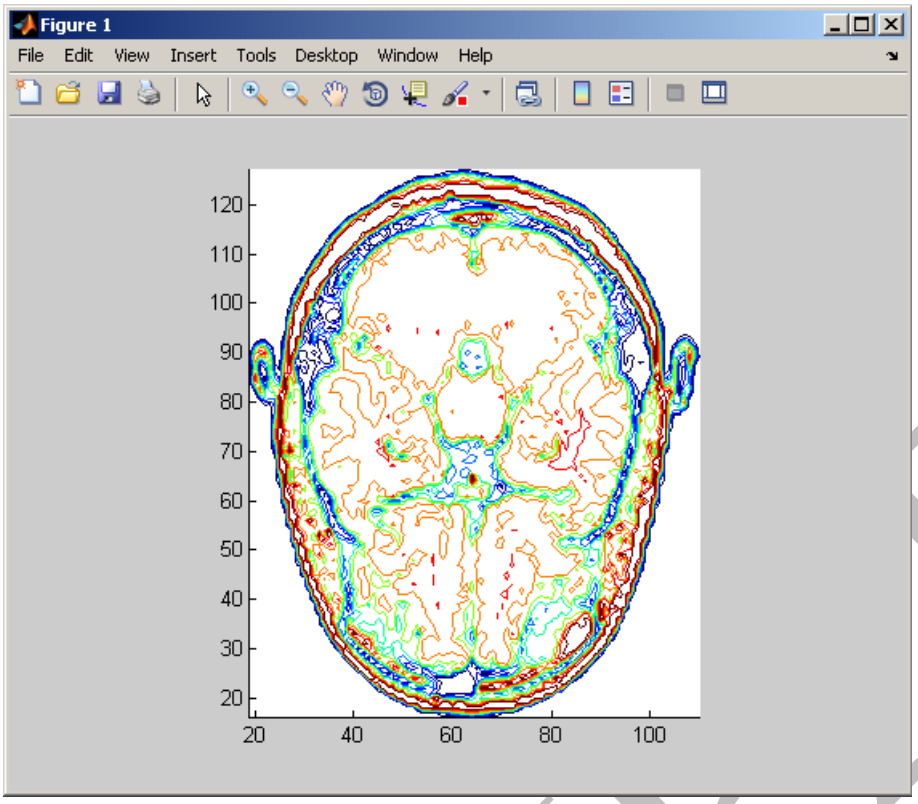
subplot (2,2,1), image(D(:,:,9))
subplot (2,2,2), image(D(:,:,1))
subplot (2,2,3), image(D(:,:,3))
subplot (2,2,4), image(D(:,:,7))
```



Now we change the aspect:

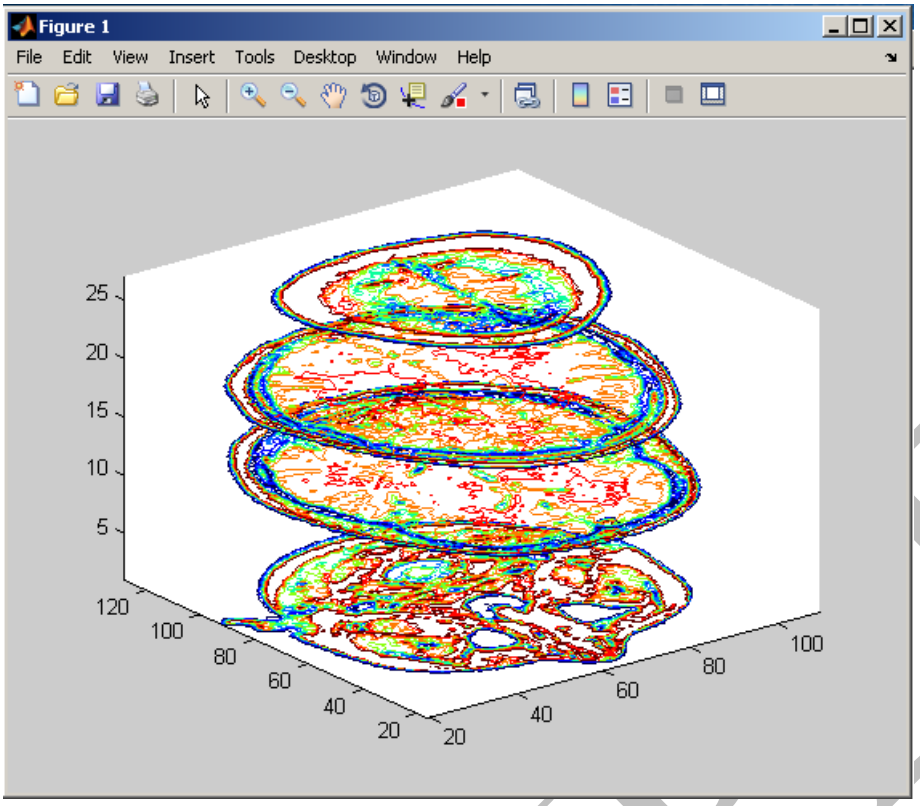
```
load mri;
D=squeeze(D);

slice_number_to_view=9;
cm = brighten(jet(length(map)),-.5);
figure('Colormap',cm);
contourslice(D,[],[],slice_number_to_view);
axis image;
```



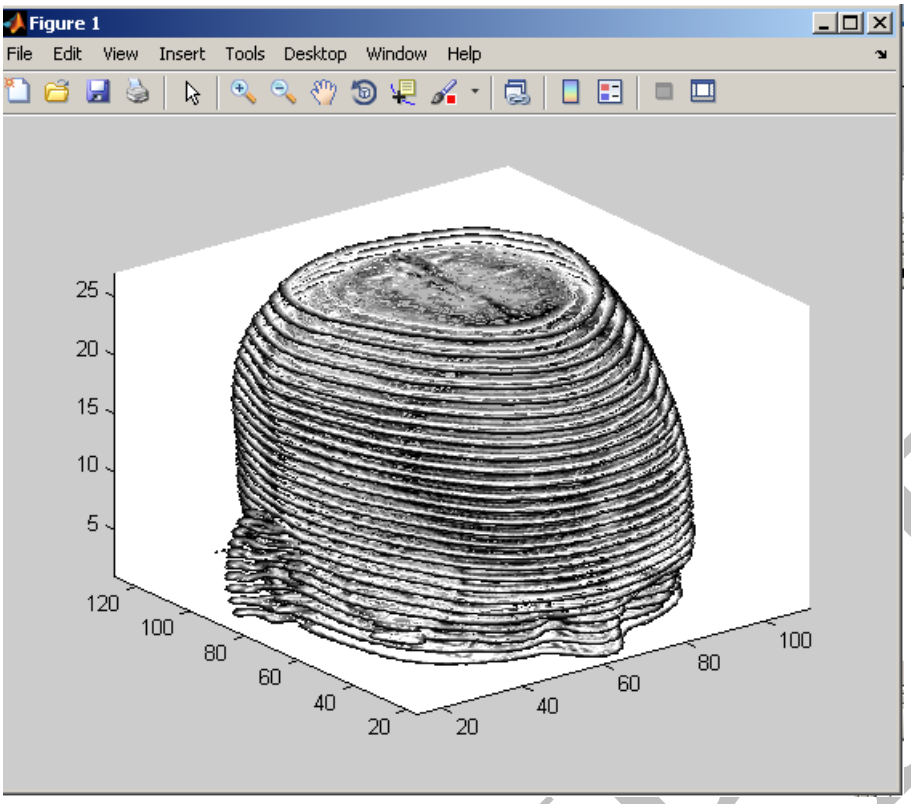
EDITOR PUBLISH VIEW

```
1    load mri;
2    D=squeeze(D);
3
4    figure('Colormap', brighten(jet(length(map)),-.5));
5    contourslice(D,[],[],[1,12,19,27],8);
6    view(3); axis tight;
```



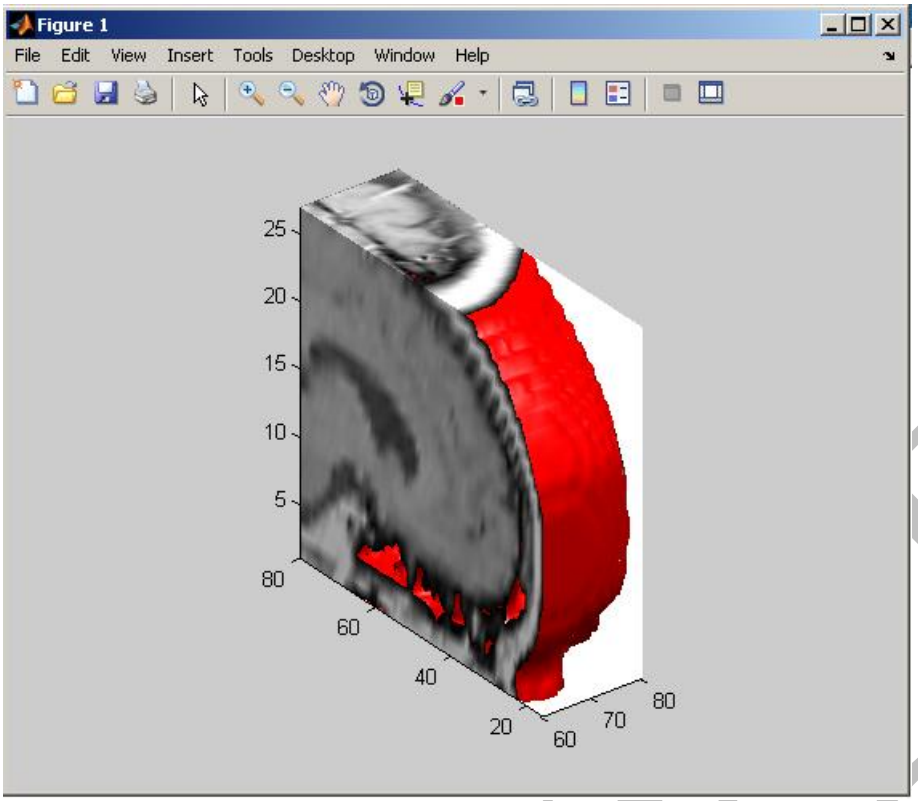
EDITOR PUBLISH VIEW

```
1     load mri;
2     D=squeeze(D);
3
4     figure('Colormap', brighten(jet(length(map)),-.5))
5     contourslice(D,[],[],1:size(D,3))
6     colormap(map), view(3), axis tight
7
```



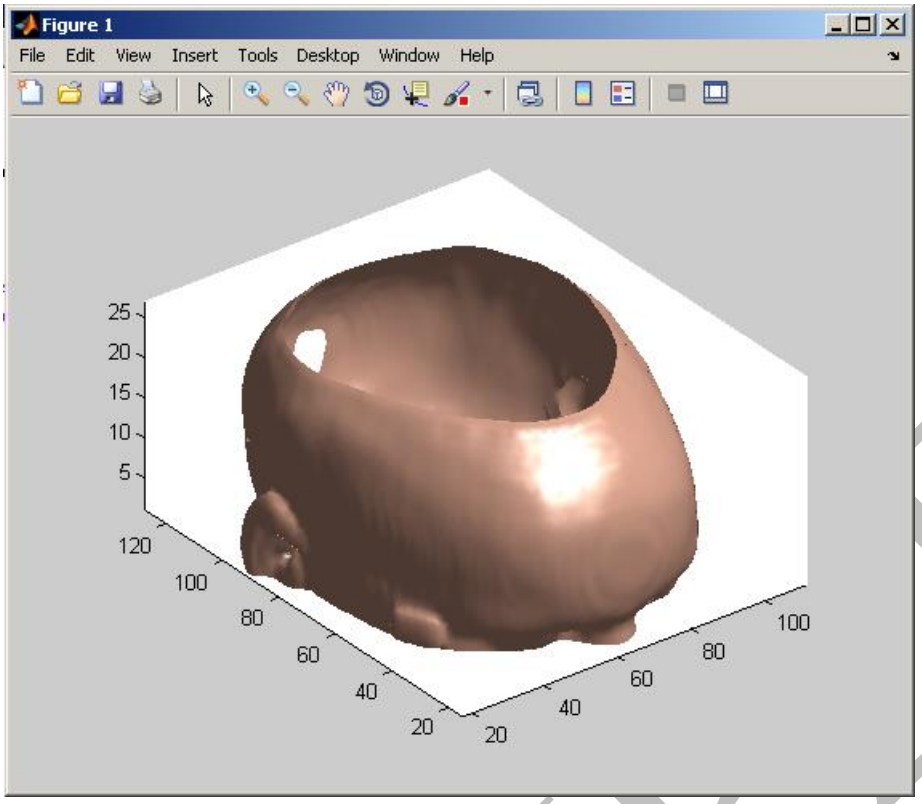
EDITOR PUBLISH VIEW

```
1    load mri;
2    D=squeeze(D);
3
4    [x,y,z,D]=subvolume(D,[60,80,nan,80,nan,nan]);
5    p1=patch(isosurface(x,y,z,D,5),'FaceColor','red','Edgecolor','none');
6    isonormals(x,y,z,D,p1);
7    p2=patch(isocaps(x,y,z,D,5),'FaceColor','interp','Edgecolor','none');
8    view(3); axis tight; daspect([1,1,.4]);colormap(gray(100))
9    camlight right;camlight left;lighting gouraud
```



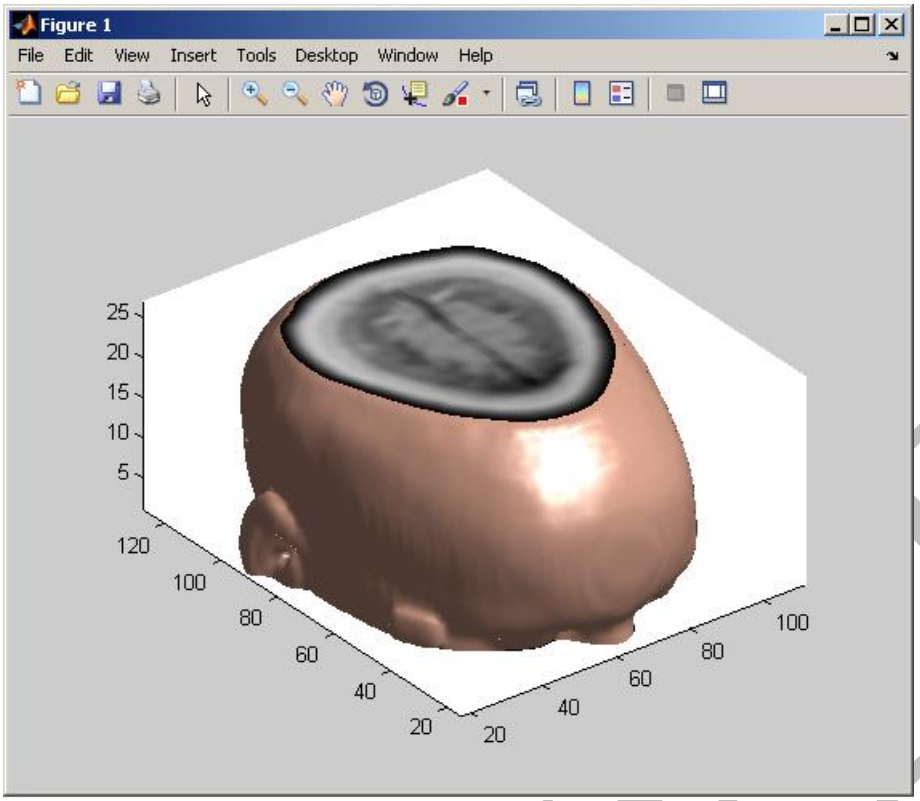
EDITOR PUBLISH VIEW

```
1     load mri;
2     D=squeeze(D);
3
4     figure('Colormap',brighten(jet(length(map)),-.5));
5     D = smooth3(D);
6     p = patch( isosurface(D,5) );
7     isonormals(D, p);
8     set(p, 'FaceColor',[1,.75,.65], 'EdgeColor','none')
9     daspect([1 1 .5]), view(3), axis tight, axis vis3d
10    camlight, lighting gouraud
```



EDITOR PUBLISH VIEW

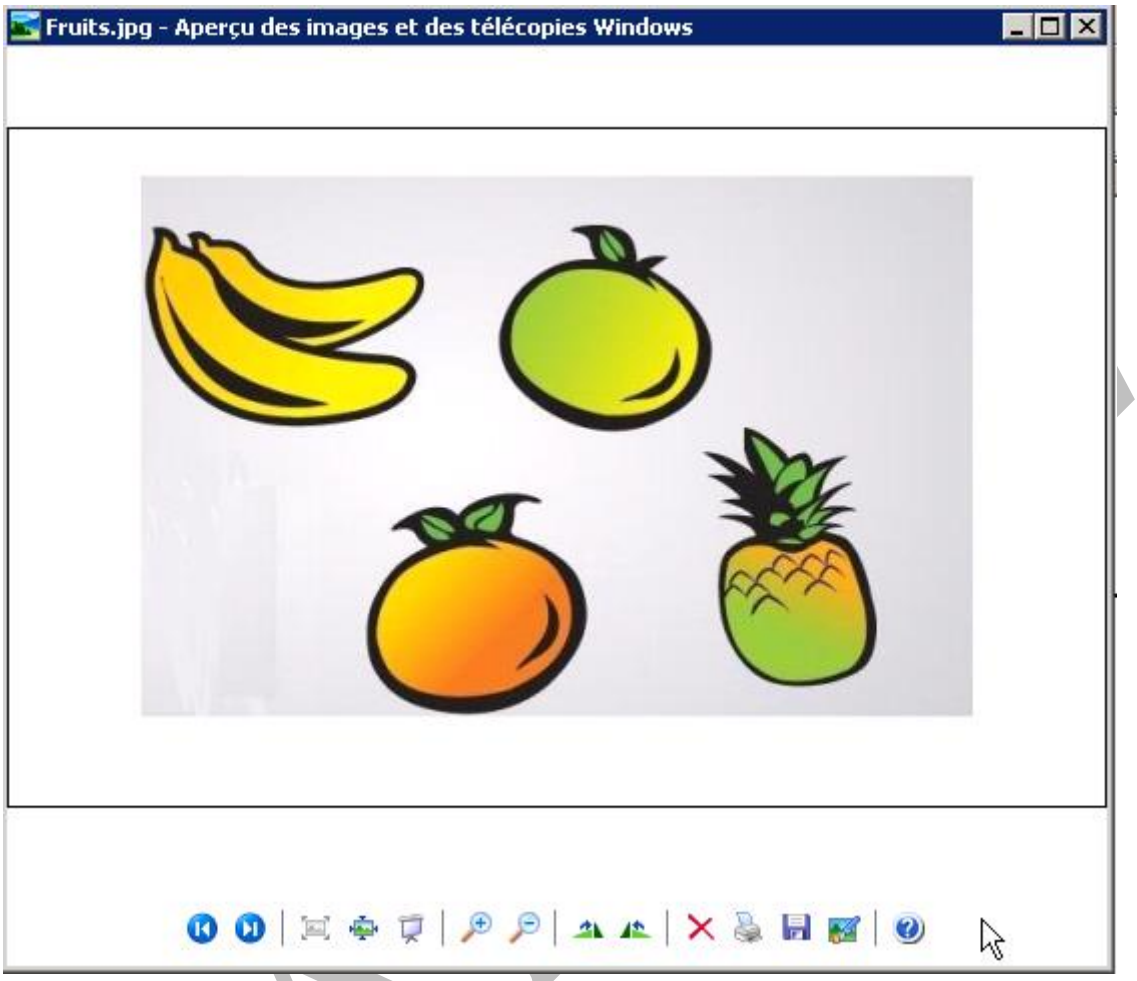
```
1   load mri;
2   D=squeeze(D);
3
4   figure('Colormap',brighten(jet(length(map)),-.5));
5   D = smooth3(D);
6   p = patch( isosurface(D,5) );
7   isonormals(D, p);
8   set(p, 'FaceColor',[1,.75,.65], 'EdgeColor','none')
9   daspect([1 1 .5]), view(3), axis tight, axis vis3d
10  camlight, lighting gouraud
11
12  patch(isocaps(D,5), 'FaceColor','interp', 'EdgeColor','none');
13  colormap(map)
```


Figure 1
File Edit View Insert Tools Desktop Window Help
25
20
15
10
5
120
100
80
60
40
20
20
40
60
80
100

SAMPLE

Objects recognition and extraction

Extraction without reference images



SA

EDITOR PUBLISH VIEW

```
I=imread('C:\tmp\Fruits.jpg');

figure(1);
imshow(I);

%Convert the Image to binary
BW=imcomplement(im2bw(I));
figure(2);
imshow(BW);

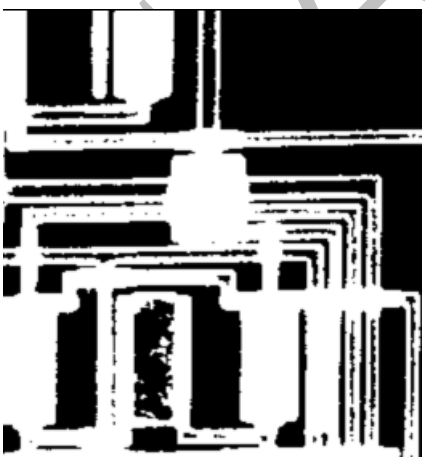
%Fill the holes
C=imfill(BW,'holes');
figure(3);
imshow(C);

%Label the connected components
[Label,Total]=bwlabel(C);

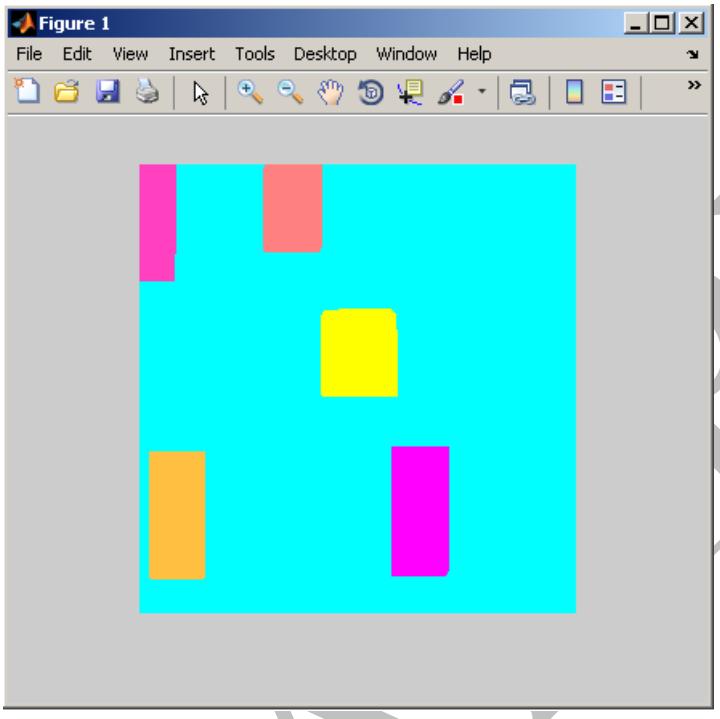
%Rectangle containing the region
Sdata=regionprops(Label,'BoundingBox');

%Crop all the Coins
for i=1:Total
    Img=imcrop(I,Sdata(i).BoundingBox);
    Name=strcat('Object Number:',num2str(i));
    figure,imshow(Img); title(Name);
end
```

Extraction with reference shape matrix

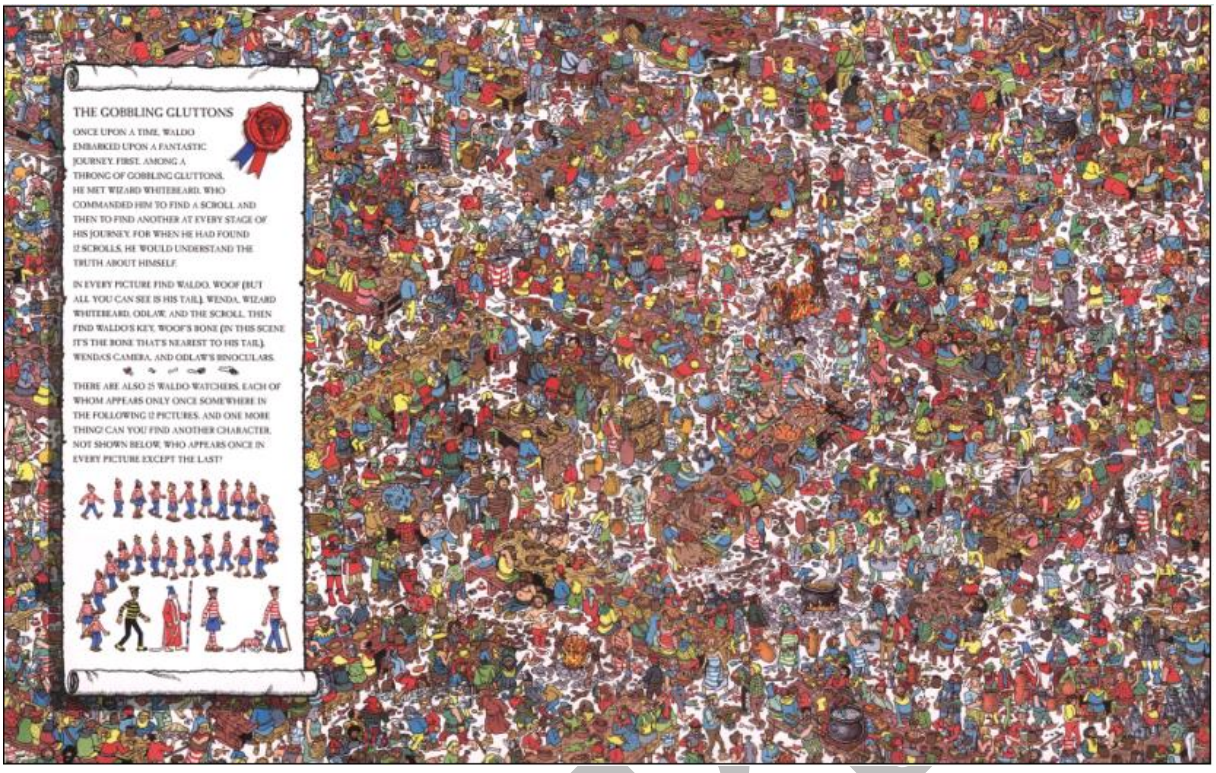


```
%morphological techniques of erosion and dilation.
BW1 = imread('circbw.tif');
SE = strel('rectangle',[40 30]);
BW2 = imerode(BW1,SE);
BW3 = imdilate(BW2,SE);
L = bwlabel(BW3);
RGB = label2rgb(L, 'spring','c', 'shuffle');
imshow(RGB);
```



Extraction with reference image

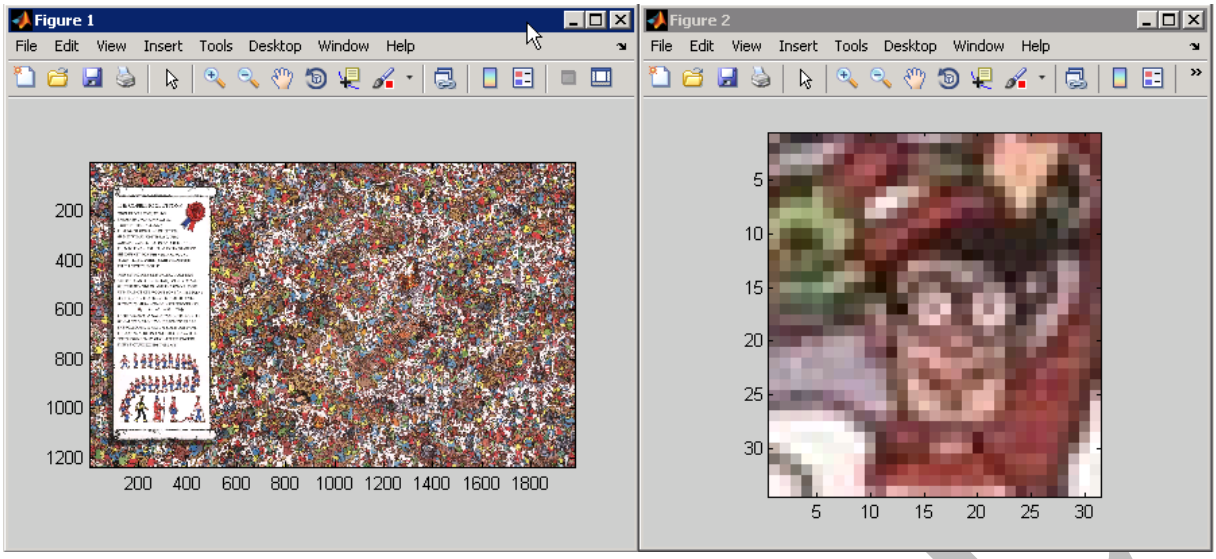
SAMPLE



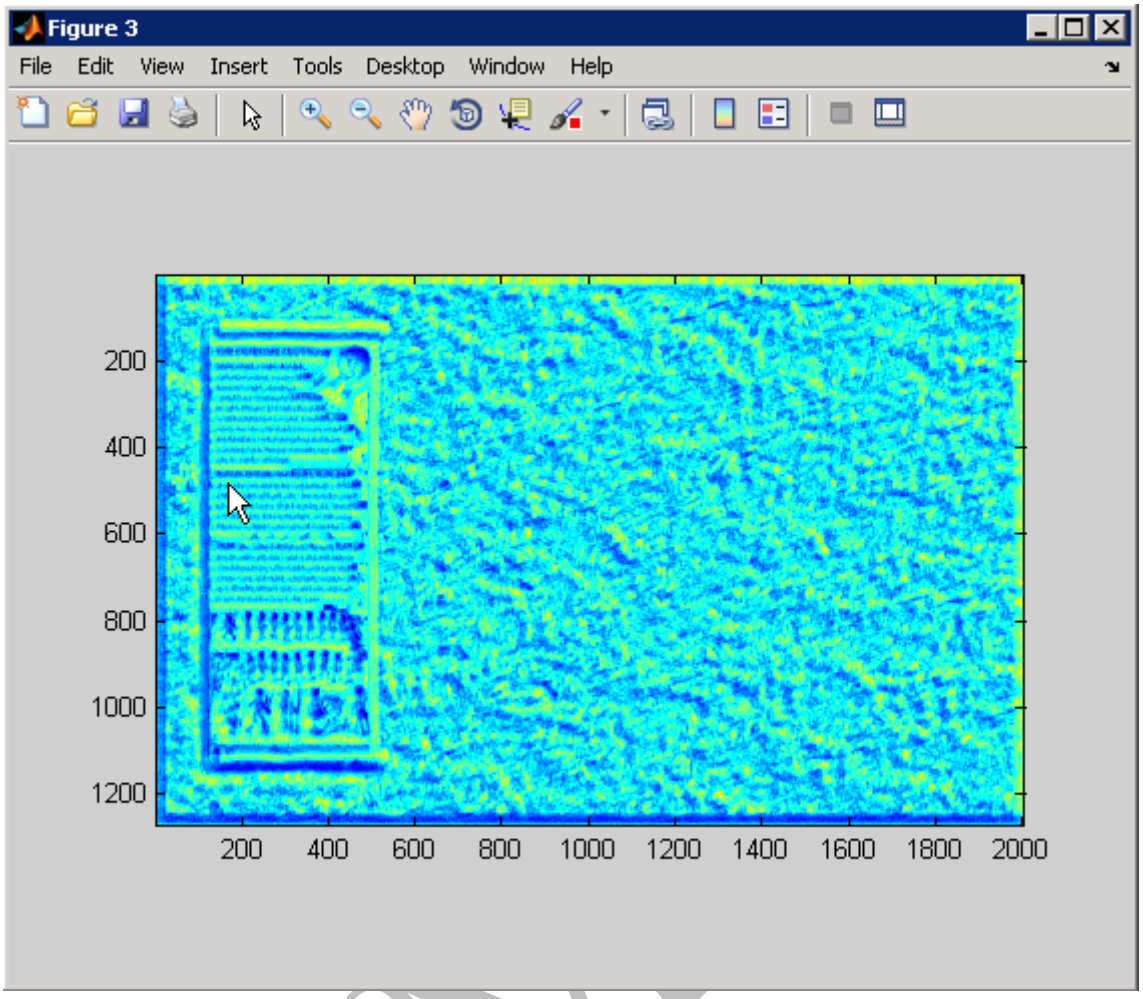


EDITOR PUBLISH VIEW

```
1 -    clear all;close all;clc
2 -    template = importdata('C:\tmp\Waldo.png');
3 -    im = importdata('C:\tmp\FindWaldo.jpg');
4
5      %we show the images
6 -    imagesc(im); axis image;
7 -    figure;
8 -    imagesc(template); axis image;
```

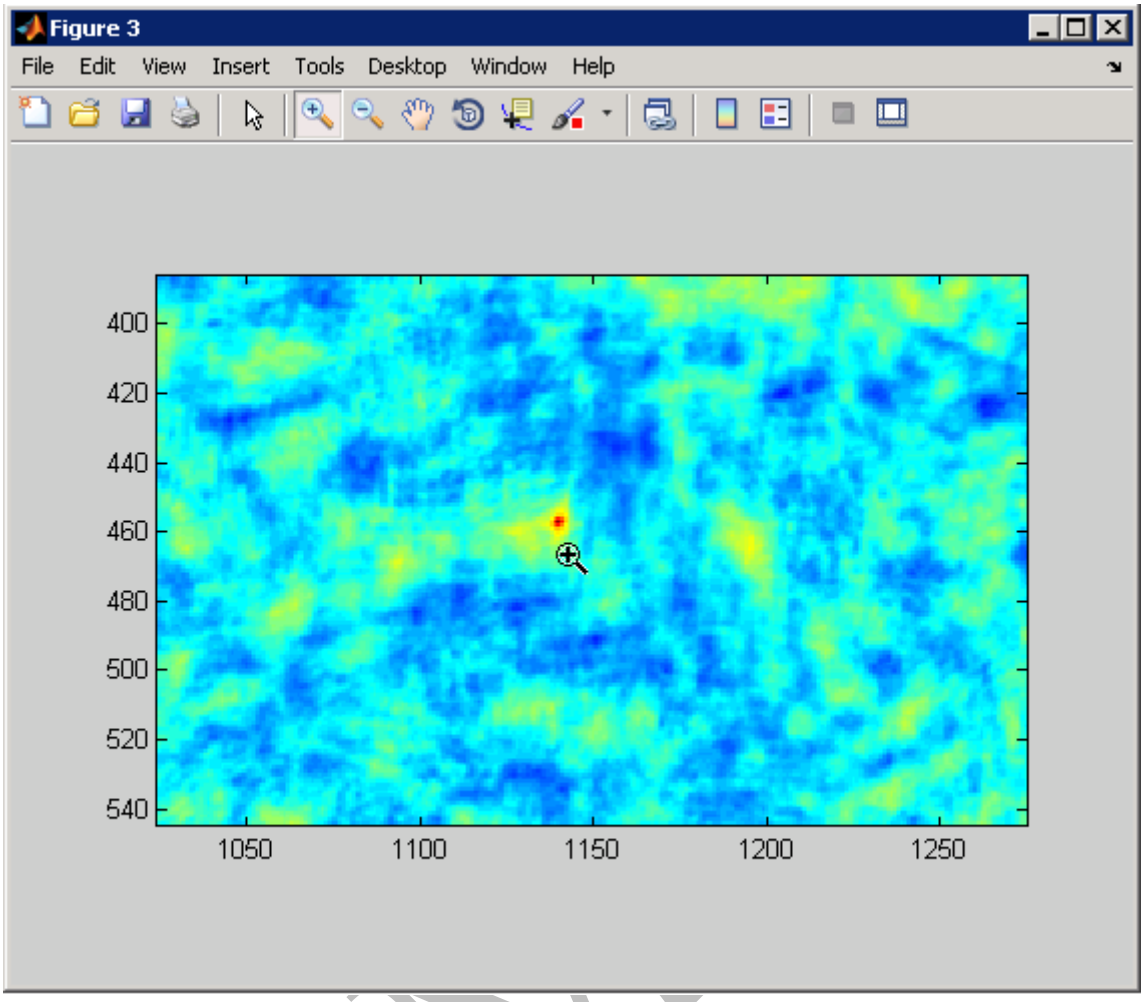



```
9
10      %we convert in grayscale
11 -    template = mean(template, 3);
12 -    im = mean(im, 3);
13
14      %search 2D correlation using the normxcorr2 function
15      %of the image processing toolbox
16 -    figure;
17 -    W = normxcorr2(template, im);
18 -    imagesc(W); axis image; colormap('jet');
19
20 -    W1 = W(:);
21 -    size(W1);
22      % we search the max correlation
23      % I gives the number of pixel (we will get the coordinates later with
24      % ind2sub() function
25 -    [C, I] = max(W1);
```



If we zoom:

SAM




```
Command Window
>> C

C =

     1.0000

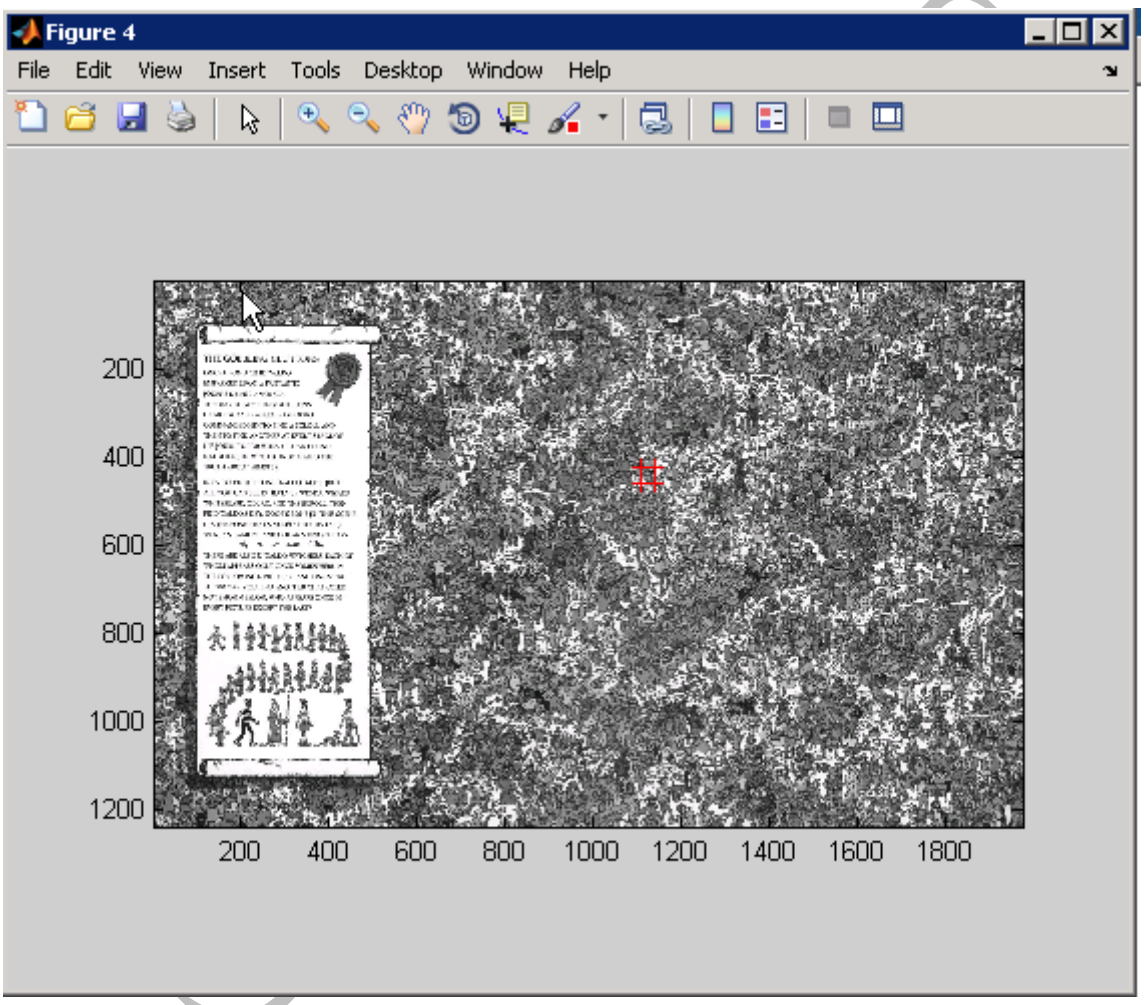
>> I

I =

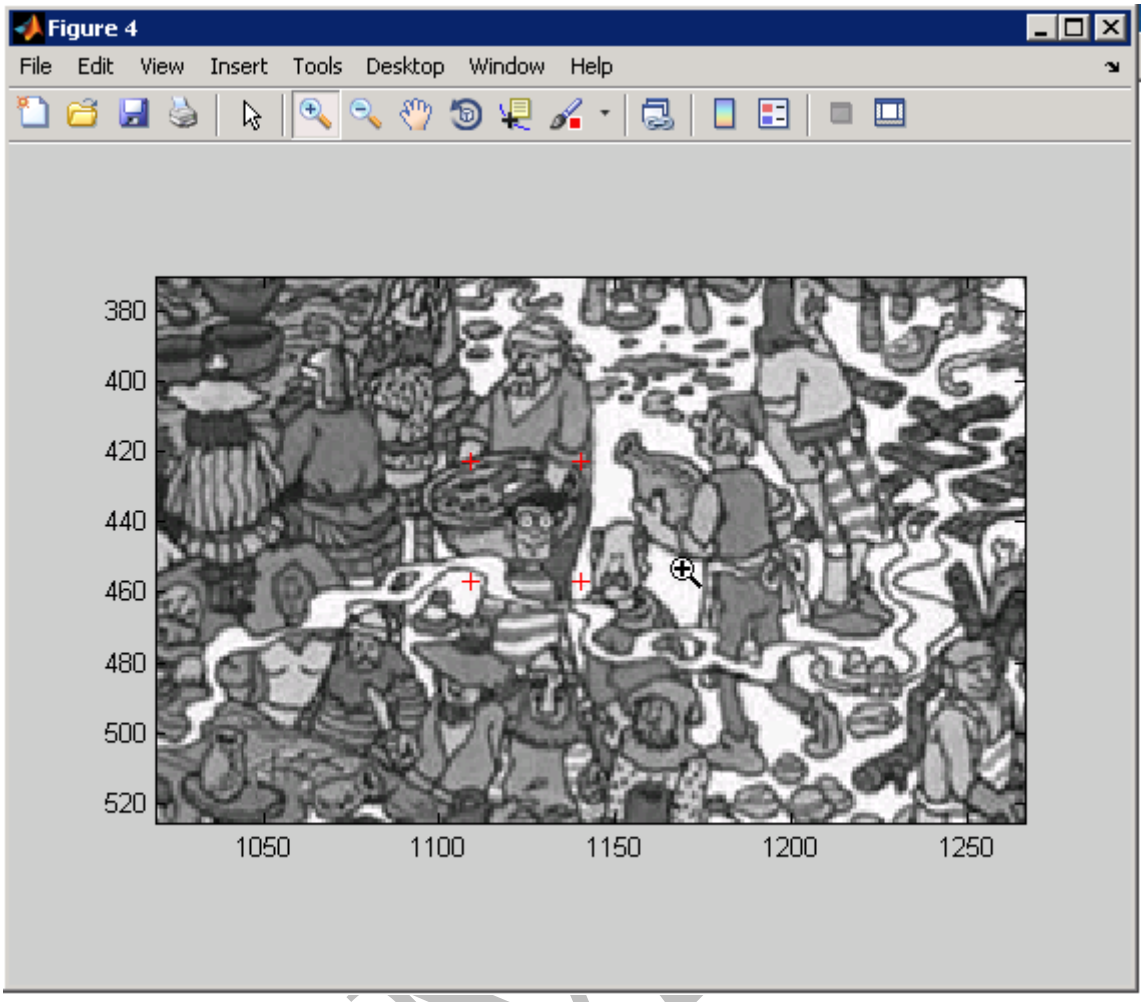
       1450404

fx >>
```

```
26
27 -    figure;
28      %we prepare the image to show where is Waldo
29 -    imagesc(im); axis image; colormap('gray');
30 -    hold on;
31
32 -    [y, x] = ind2sub(size(W), I)
33 -    scatter(x, y, 'r+');
34 -    scatter(x - size(template, 2), y, 'r+');
35 -    scatter(x, y - size(template, 1), 'r+');
36 -    scatter(x - size(template, 2), y - size(template, 1), 'r+');
```



if we zoom:



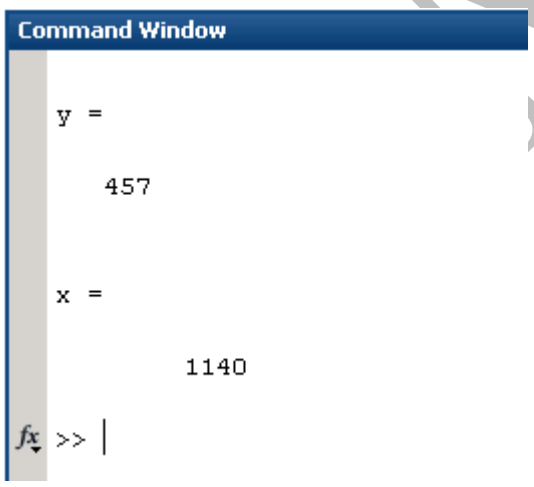


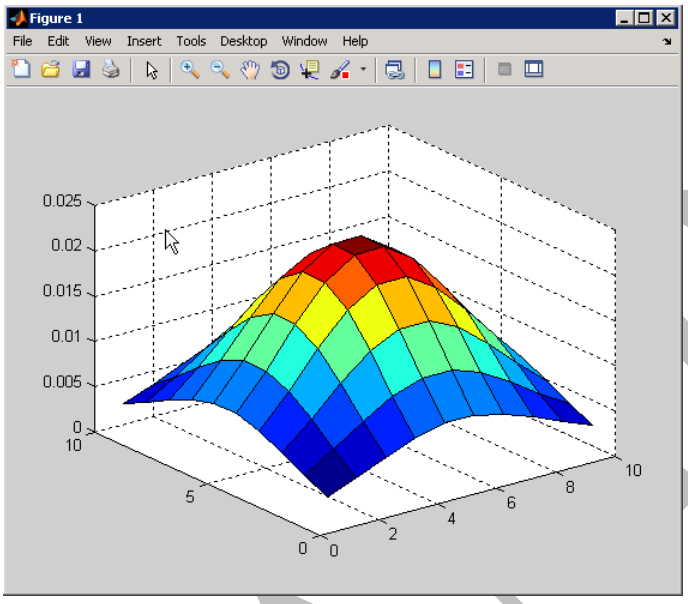
Image filters (convolution)

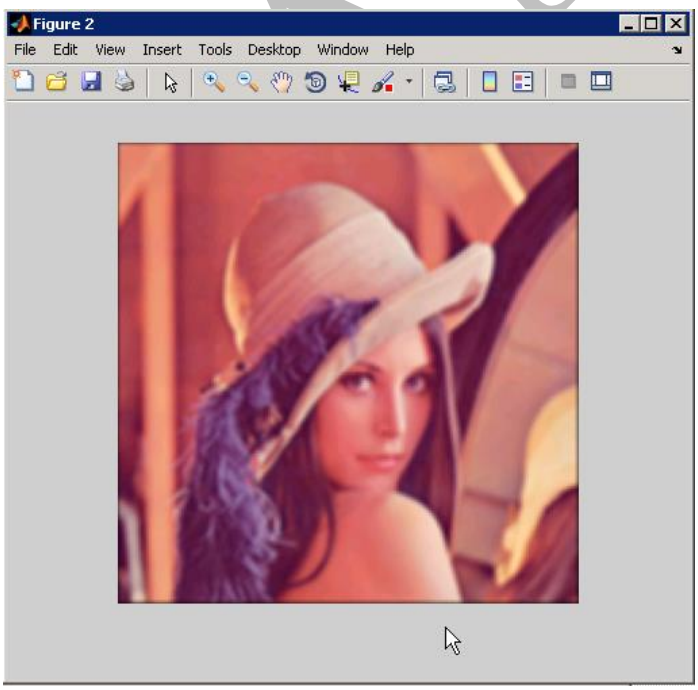
Gaussian (blur) filter

1st method using imfilter()

EDITOR PUBLISH VIEW

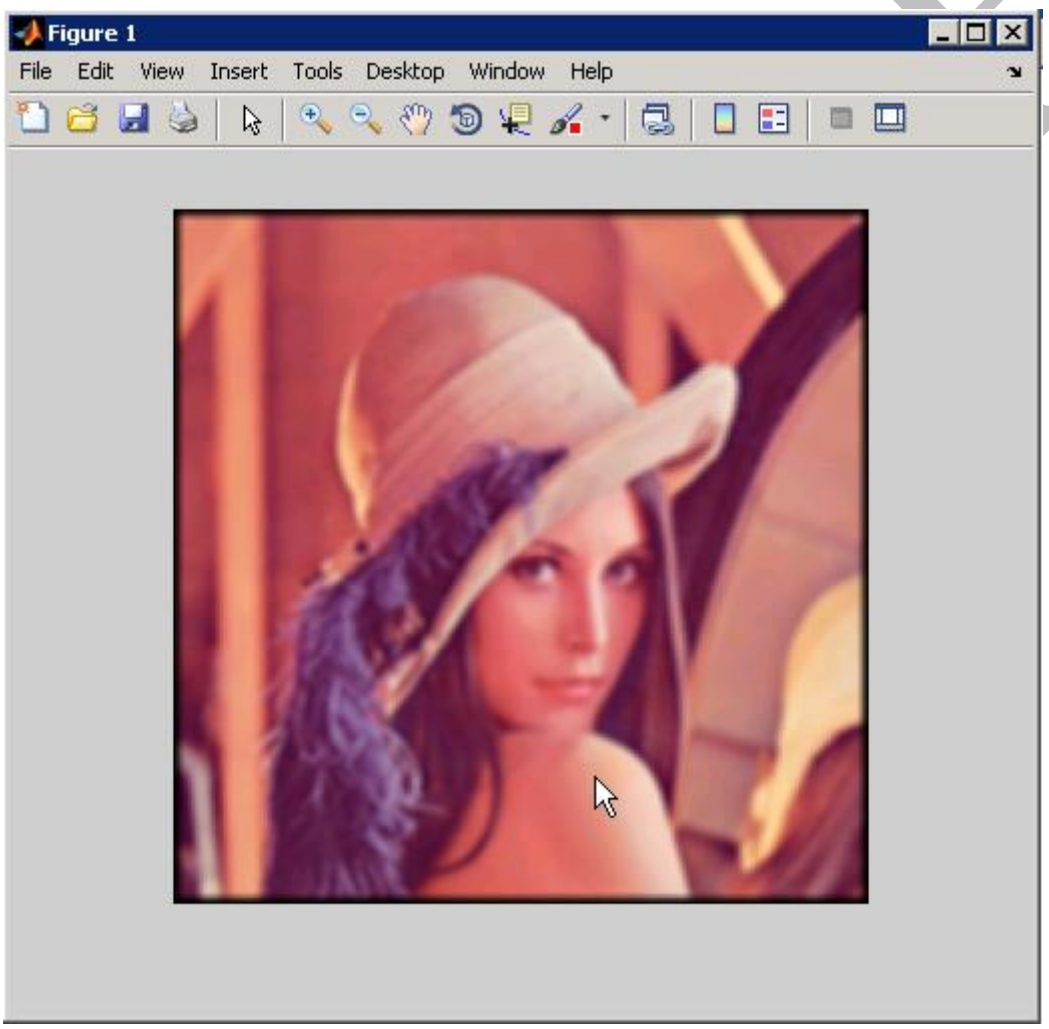
```
1 -    clear all;close all;clc
2 -    img=imread('C:/tmp/Lenna.png');
3 -    img=im2double(img);
4 -    matrix_conv_size=10;
5 -    sigma=3;
6 -    filter=fspecial('gaussian',matrix_conv_size,sigma);
7 -    surf(filter);
8 -    conv=imfilter(img,filter);
9      figure
10 -   imshow(conv)
```





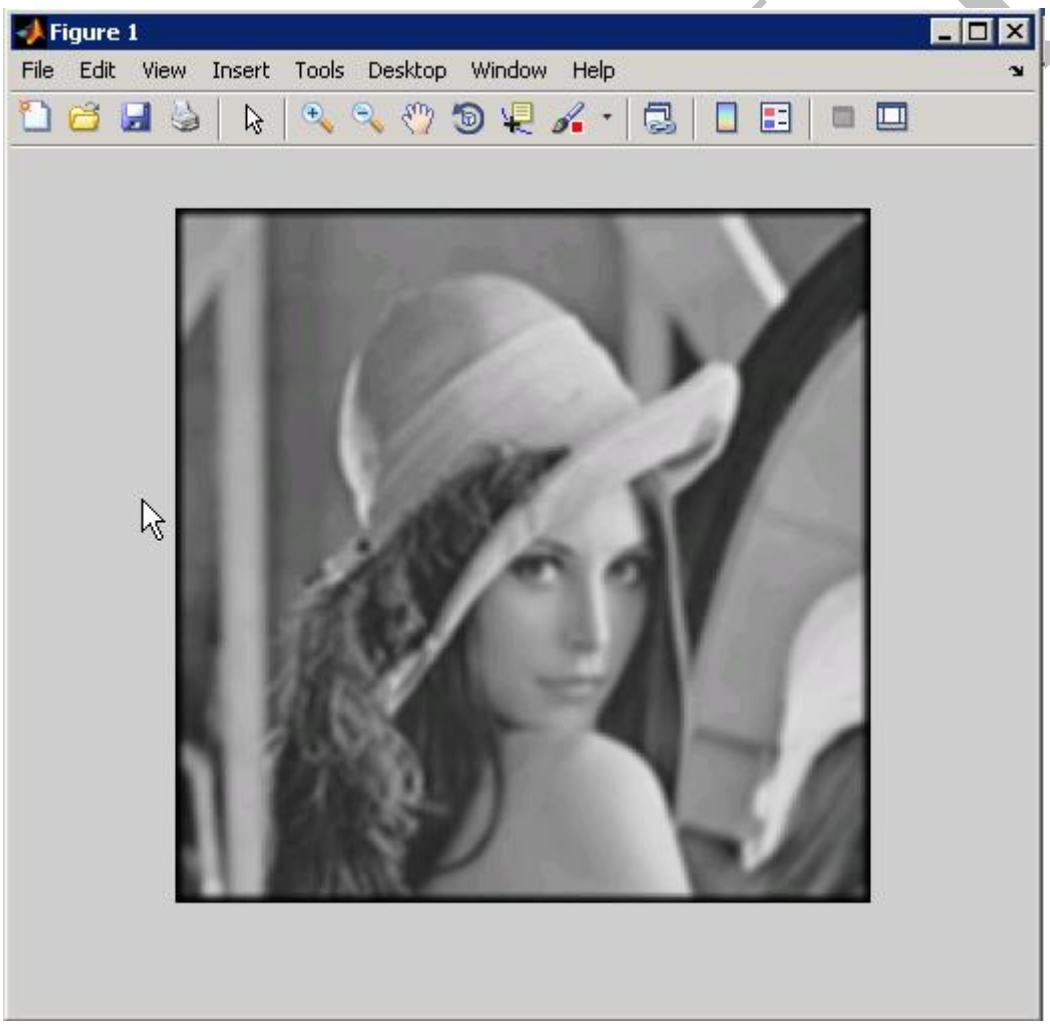
2nd method using convn()

```
1     %Color images are 3 dimensional arrays (x,y,color).
2     %conv2() is only defined for 2-dimensions, so it won't work directly on a 3D array.
3     %Solution: use convn
4     clear all;close all;clc
5     img=imread('C:/tmp/Lenna.png');
6     img=im2double(img);
7     matrix_conv_size=10;
8     sigma=3;
9     filter=fspecial('gaussian',matrix_conv_size,sigma);
10    conv=convn(img,filter);
11    imshow(conv)
```

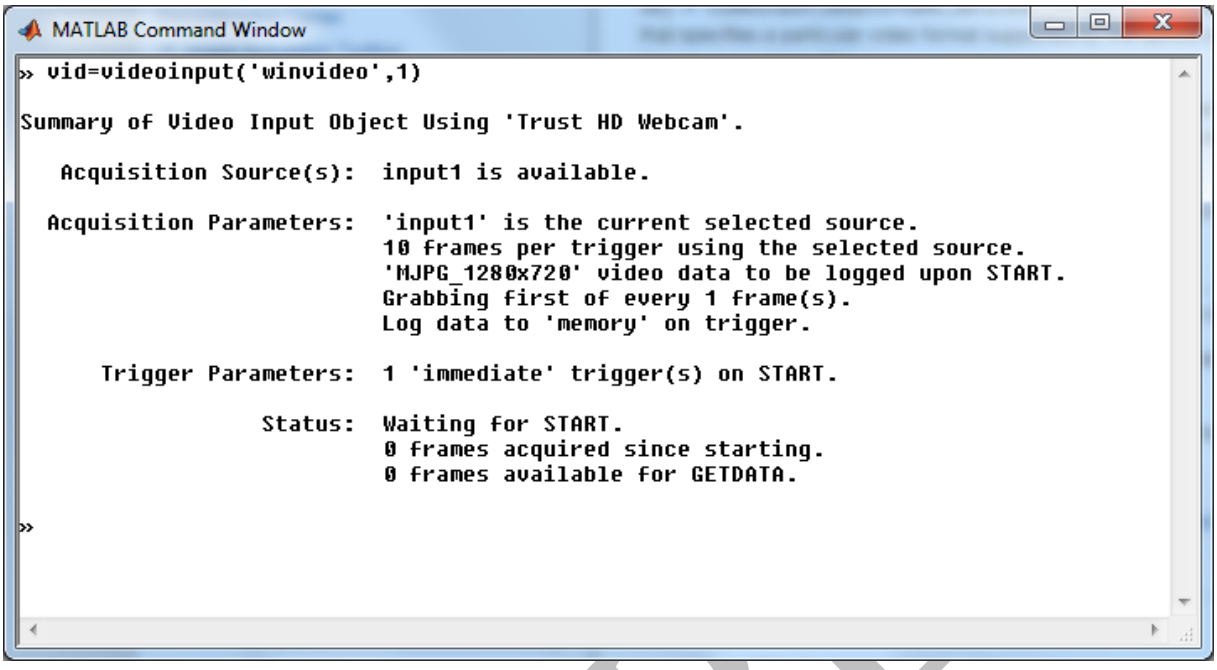


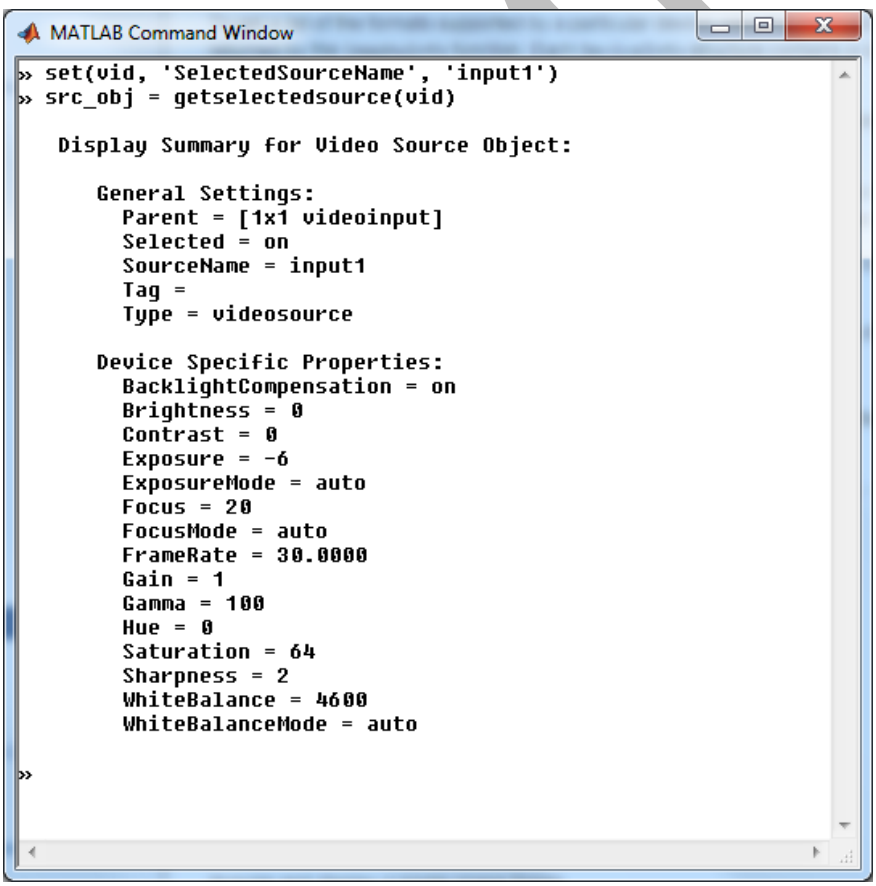
3rd method using conv2()

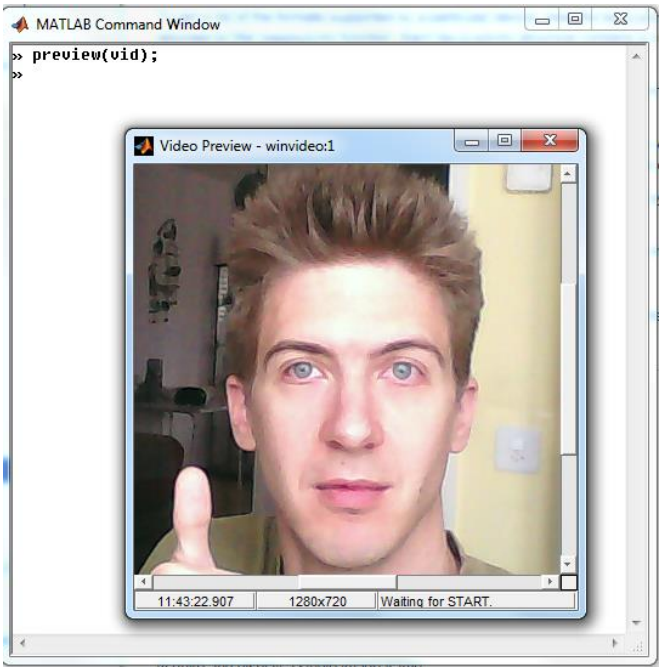
```
1   %conv2() is only defined for 2-dimensions so we have to work
2   %on a grayscale image
3   clear all;close all;clc
4   img=imread('C:/tmp/Lenna.png');
5   img=rgb2gray(img); %img=uint8(repmat(mean(img,3),[1,1,3]));
6   matrix_conv_size=10;
7   sigma=3;
8   filter=fspecial('gaussian',matrix_conv_size,sigma);
9   conv=conv2(img,filter);
10  imshow(uint8(conv));
```

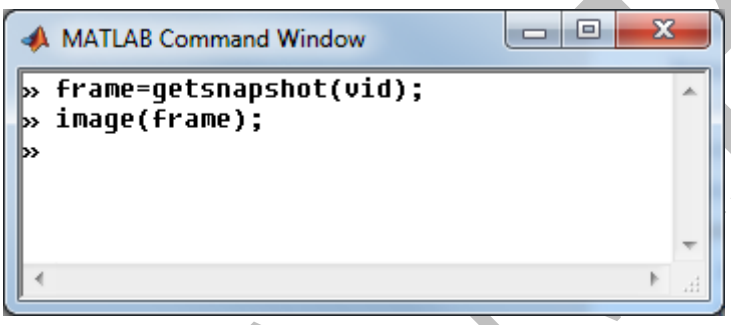


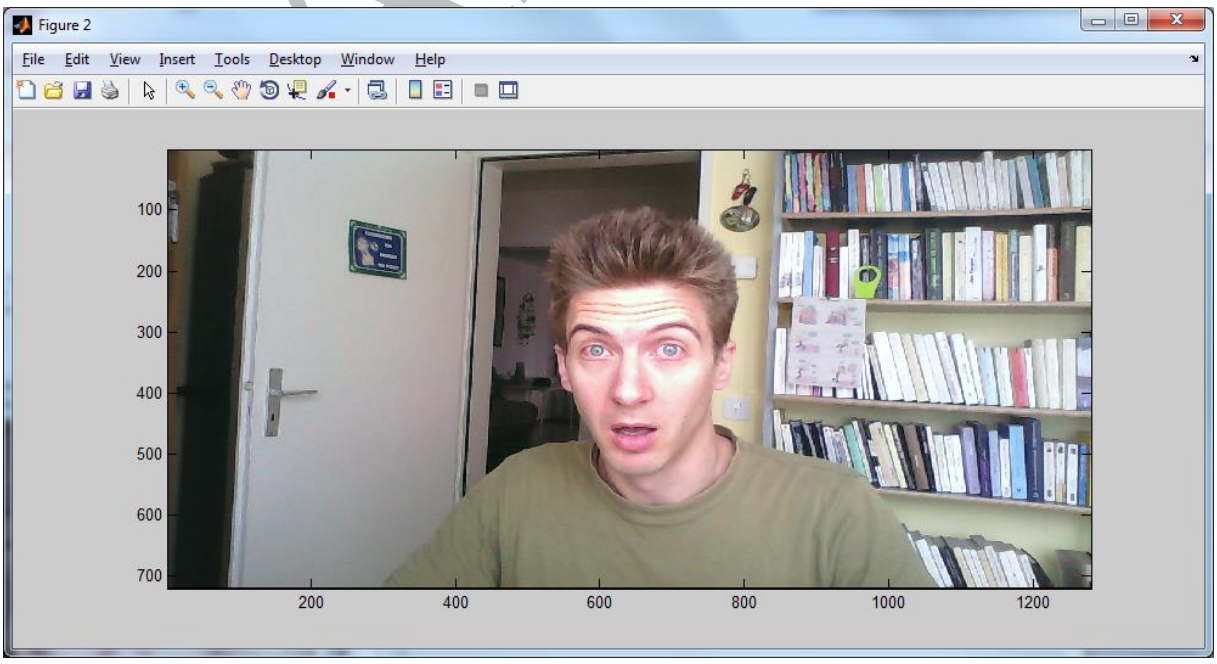
External Camera Acquisition



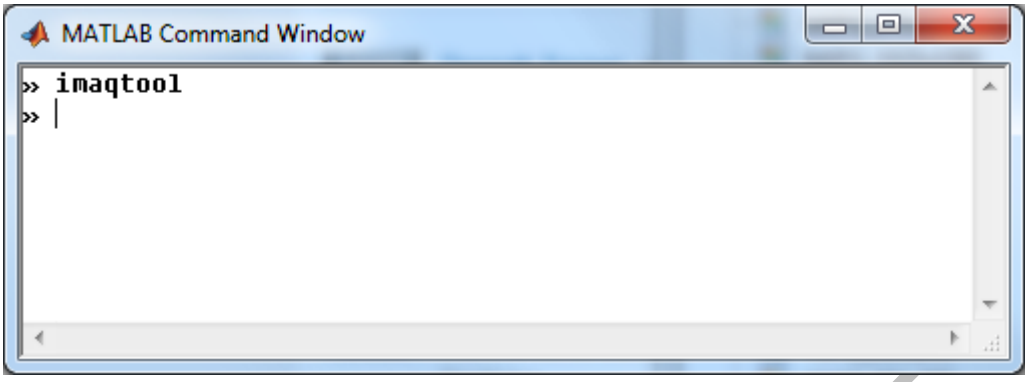


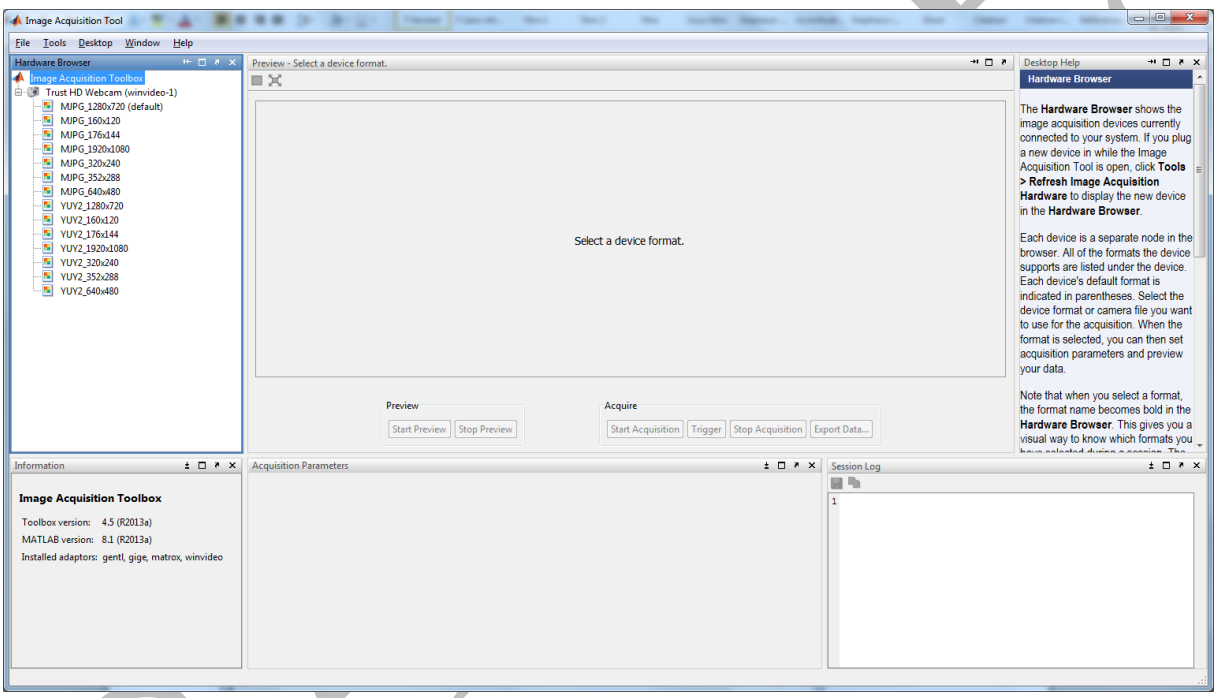

MATLAB Command Window
» preview(vid);
»
Video Preview - winvideo:1
11:43:22.907
1280x720
Waiting for START.

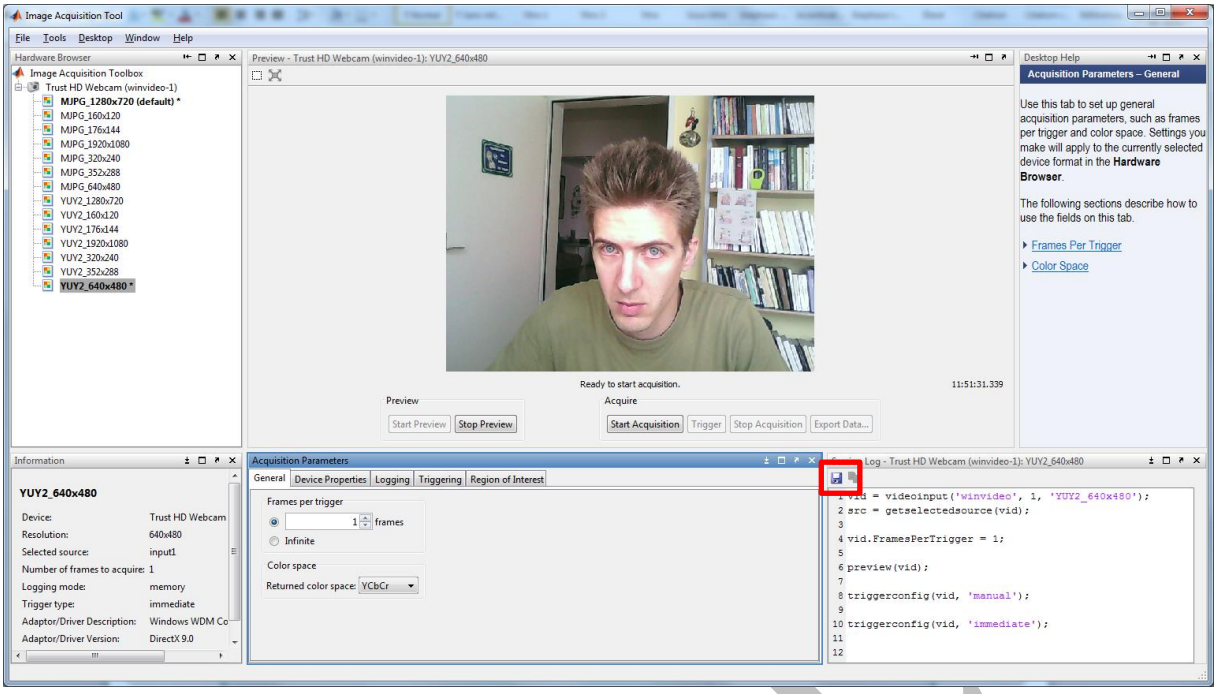

MATLAB Command Window
» frame=getsnapshot(vid);
» image(frame);
»


Figure 2
File Edit View Insert Tools Desktop Window Help
100
200
300
400
500
600
700
200
400
600
800
1000
1200

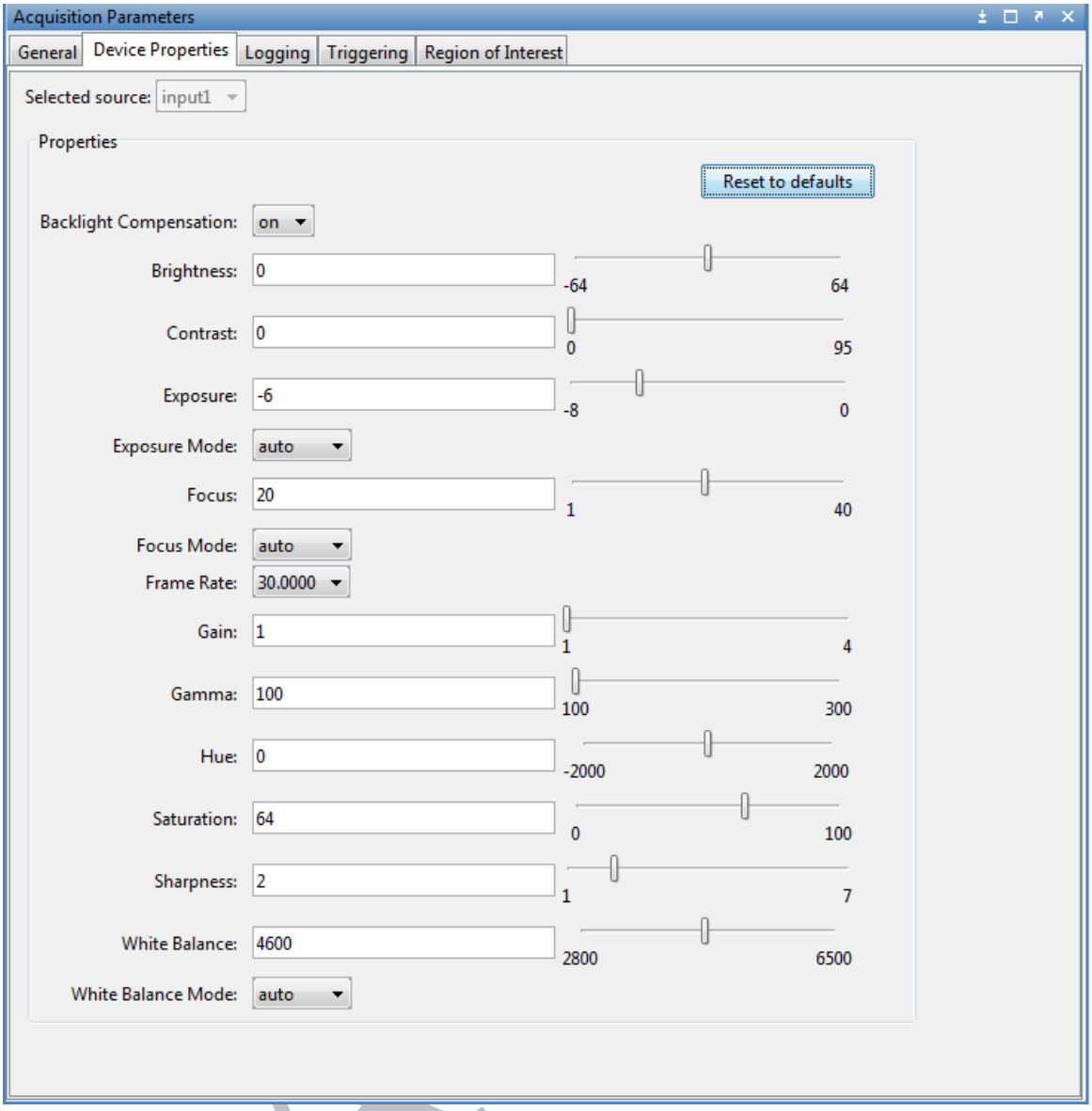
SAMPLE

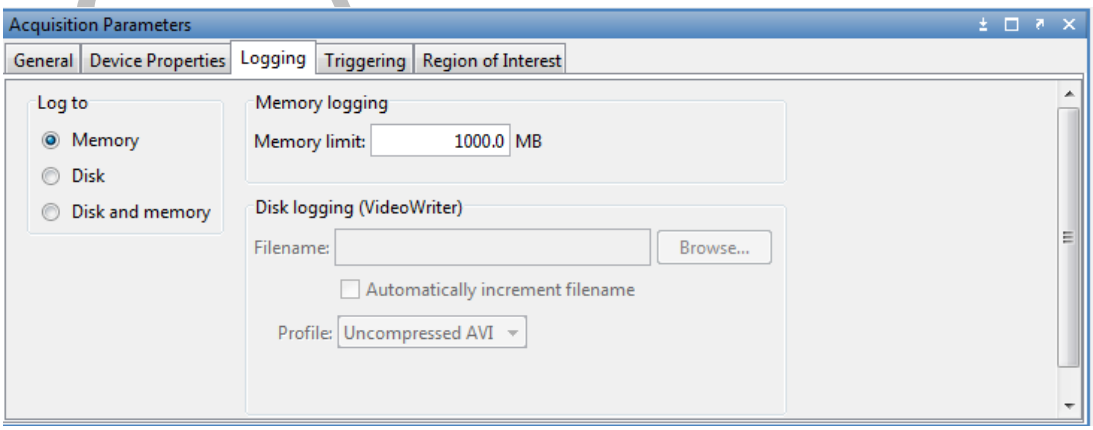

MATLAB Command Window
» imaqtool
»

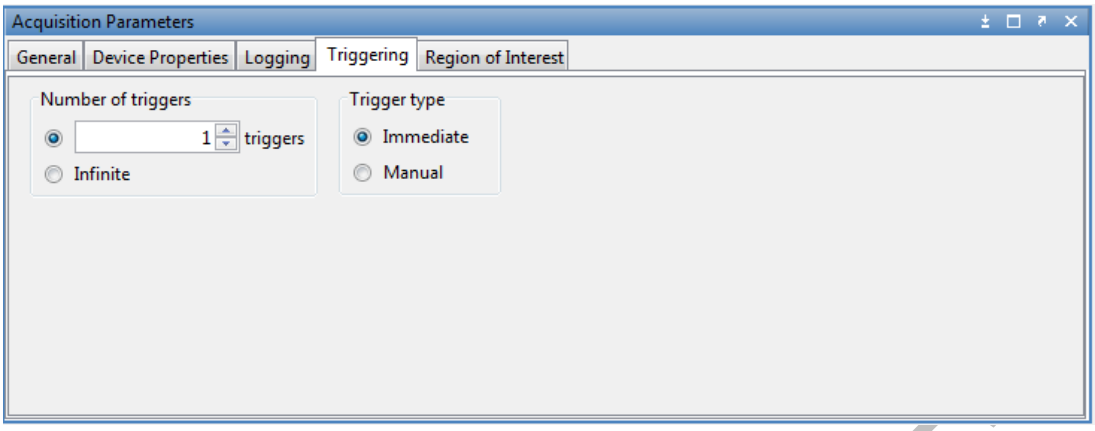

Image Acquisition Tool
File Tools Desktop Window Help
Hardware Browser
Image Acquisition Toolbox
Trust HD Webcam (winvideo-1)
MJPG_1280x720 (default)
MJPG_160x120
MJPG_176x144
MJPG_1920x1080
MJPG_320x240
MJPG_352x288
MJPG_640x480
YUY2_1280x720
YUY2_160x120
YUY2_176x144
YUY2_1920x1080
YUY2_320x240
YUY2_352x288
YUY2_640x480
Preview - Select a device format.
Select a device format.
Preview
Start Preview
Stop Preview
Acquire
Start Acquisition
Trigger
Stop Acquisition
Export Data...
Desktop Help
Hardware Browser
The Hardware Browser shows the image acquisition devices currently connected to your system. If you plug a new device in while the Image Acquisition Tool is open, click Tools > Refresh Image Acquisition Hardware to display the new device in the Hardware Browser.
Each device is a separate node in the browser. All of the formats the device supports are listed under the device. Each device's default format is indicated in parentheses. Select the device format or camera file you want to use for the acquisition. When the format is selected, you can then set acquisition parameters and preview your data.
Note that when you select a format, the format name becomes bold in the Hardware Browser. This gives you a visual way to know which formats you
Information
Image Acquisition Toolbox
Toolbox version: 4.5 (R2013a)
MATLAB version: 8.1 (R2013a)
Installed adaptors: gentl, gige, matrox, winvideo
Acquisition Parameters
Session Log
1

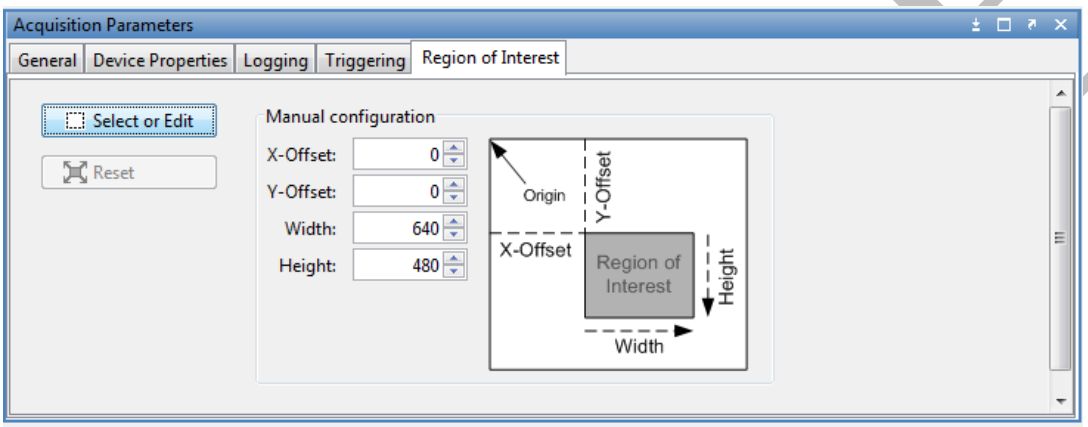

Image Acquisition Tool
File Tools Desktop Window Help
Hardware Browser
Image Acquisition Toolbox
Trust HD Webcam (winvideo-1)
MJPG_1280x720 (default) *
MJPG_160x120
MJPG_176x144
MJPG_1920x1080
MJPG_320x240
MJPG_352x288
MJPG_640x480
YUY2_1280x720
YUY2_160x120
YUY2_176x144
YUY2_1920x1080
YUY2_320x240
YUY2_352x288
YUY2_640x480 *
Preview - Trust HD Webcam (winvideo-1): YUY2_640x480
Ready to start acquisition.
11:51:31.339
Preview
Start Preview
Stop Preview
Acquire
Start Acquisition
Trigger
Stop Acquisition
Export Data...
Desktop Help
Acquisition Parameters – General
Use this tab to set up general acquisition parameters, such as frames per trigger and color space. Settings you make will apply to the currently selected device format in the Hardware Browser.
The following sections describe how to use the fields on this tab.
Frames Per Trigger
Color Space
Information
YUY2_640x480
Device: Trust HD Webcam
Resolution: 640x480
Selected source: input1
Number of frames to acquire: 1
Logging mode: memory
Trigger type: immediate
Adaptor/Driver Description: Windows WDM Co
Adaptor/Driver Version: DirectX 9.0
Acquisition Parameters
General
Device Properties
Logging
Triggering
Region of Interest
Frames per trigger
1 frames
Infinite
Color space
Returned color space: YCbCr
Log - Trust HD Webcam (winvideo-1): YUY2_640x480
vid = videoinput('winvideo', 1, 'YUY2_640x480');
src = getselectedsource(vid);
vid.FramesPerTrigger = 1;
preview(vid);
triggerconfig(vid, 'manual');
triggerconfig(vid, 'immediate');

SAMPLE


Acquisition Parameters
General
Device Properties
Logging
Triggering
Region of Interest
Selected source: input1
Properties
Reset to defaults
Backlight Compensation: on
Brightness: 0 -64 64
Contrast: 0 0 95
Exposure: -6 -8 0
Exposure Mode: auto
Focus: 20 1 40
Focus Mode: auto
Frame Rate: 30.0000
Gain: 1 1 4
Gamma: 100 100 300
Hue: 0 -2000 2000
Saturation: 64 0 100
Sharpness: 2 1 7
White Balance: 4600 2800 6500
White Balance Mode: auto


Acquisition Parameters
General
Device Properties
Logging
Triggering
Region of Interest
Log to
Memory
Disk
Disk and memory
Memory logging
Memory limit: 1000.0 MB
Disk logging (VideoWriter)
Filename:
Browse...
Automatically increment filename
Profile: Uncompressed AVI


Acquisition Parameters
General
Device Properties
Logging
Triggering
Region of Interest
Number of triggers
1
triggers
Infinite
Trigger type
Immediate
Manual


Acquisition Parameters
General
Device Properties
Logging
Triggering
Region of Interest
Select or Edit
Reset
Manual configuration
X-Offset:
0
Y-Offset:
0
Width:
640
Height:
480
Origin
Y-Offset
X-Offset
Region of Interest
Height
Width

SAMPLE

Bioinformatic (with Bioinformatics and Statistics Toolbox)

The Bioinformatics Toolbox extends MATLAB to provide an integrated and extendable software environment for genome and proteome analysis. Together, MATLAB and the Bioinformatics Toolbox give scientists and engineers a set of computational tools to solve problems and build applications in drug discovery, genetic engineering and biological research.

You can use the basic bioinformatic functions provided with this toolbox to create more complex algorithms and applications. These robust and well tested functions are the functions that you would otherwise have to create yourself.

- Data formats and databases — Connect to Web accessible databases. Read and convert between multiple data formats.
- Sequence analysis — Determine statistical characteristics of data. Manipulate and align sequences. Model patterns in biological sequences using Hidden Markov Model (HMM) profiles.
- Phylogenetic analysis — Create and manipulate phylogenetic tree data.
- Microarray data analysis — Read, normalize, and visualize microarray data.
- Mass spectrometry data analysis — Analyze and enhance raw mass spectrometry data.
- Statistical Learning — Classify and identify features in data sets with statistical learning tools

Complete missing values

It is quite strange that the method of k-NN is implemented only in the bioinformatics toolbox to impute missing values in a dataset, it should also be in the Financial Toolbox and the Statistical and Learning Toolbox but nevermind…

So first here are the data with the ad-hoc transformation with will deal with before using the k-NN:

SAMPLE

MATLAB Command Window

```
» mydata=dataset('File','c:/tmp/KnnImpute.csv','Delimiter',';');
» mydata=dataset2table(mydata)

mydata =

  15×4 table

    Fleur    LongueurSepale    LargeurSepale    LongueurPetale
    _____    ______________    _____________    ______________

     1       5.1               NaN              1.4
     2       4.9                 3              1.4
     3       4.7               3.2              1.3
     4       4.6               3.1              1.5
     5         5               3.6              NaN
     6         7               3.2              4.7
     7       6.4               3.2              4.5
     8       6.9               3.1              4.9
     9       5.5               2.3                4
    10       NaN               2.8              4.6
    11       6.3               3.3                6
    12       5.8               2.7              5.1
    13       7.1                 3              NaN
    14       6.3               2.9              5.6
    15       6.5                 3              5.8

» mydata=(table2cell(mydata))

mydata =

  15×4 cell array

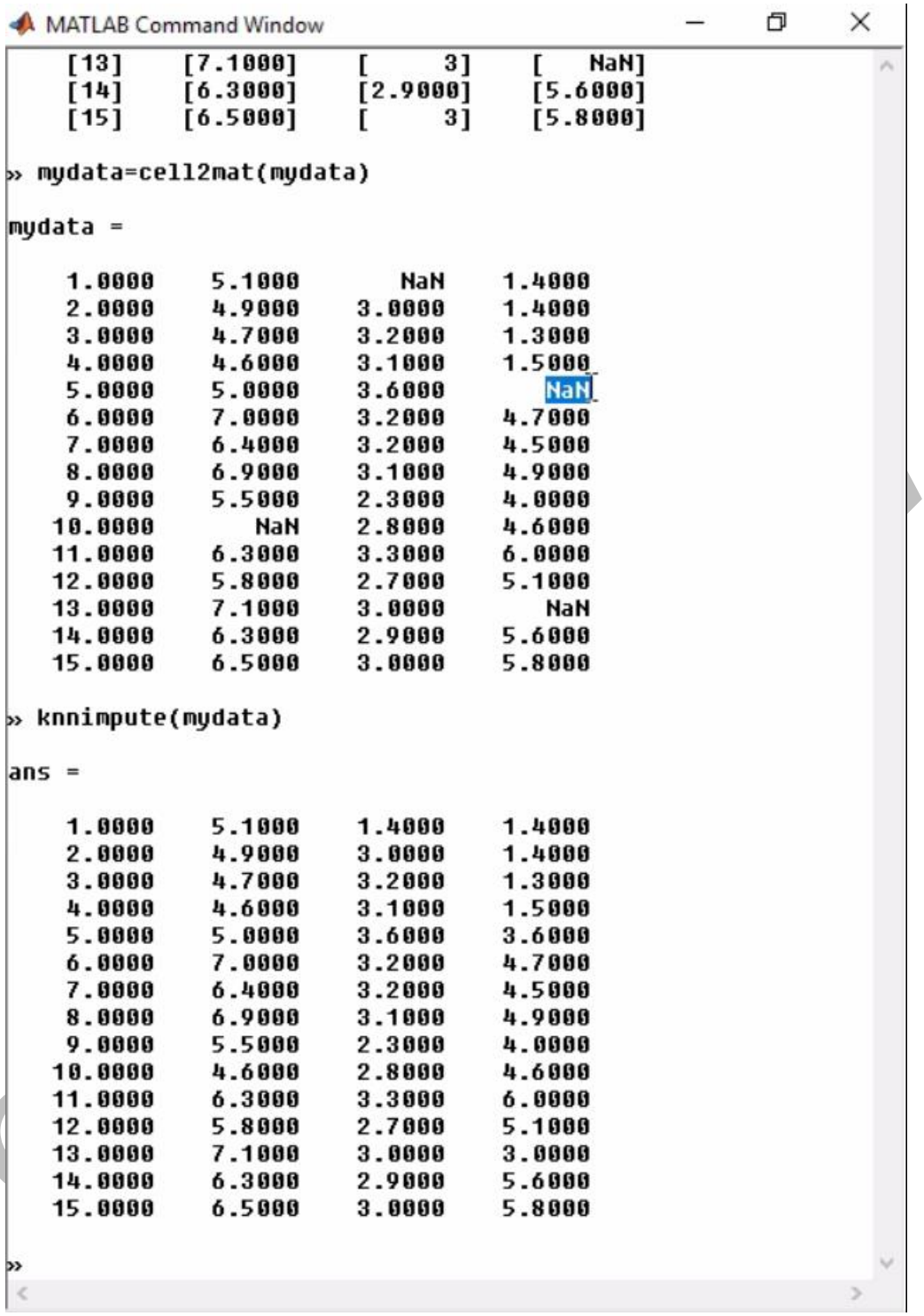
    [ 1]    [5.1000]    [   NaN]    [1.4000]
    [ 2]    [4.9000]    [     3]    [1.4000]
    [ 3]    [4.7000]    [3.2000]    [1.3000]
    [ 4]    [4.6000]    [3.1000]    [1.5000]
    [ 5]    [     5]    [3.6000]    [   NaN]
    [ 6]    [     7]    [3.2000]    [4.7000]
    [ 7]    [6.4000]    [3.2000]    [4.5000]
    [ 8]    [6.9000]    [3.1000]    [4.9000]
    [ 9]    [5.5000]    [2.3000]    [     4]
    [10]    [   NaN]    [2.8000]    [4.6000]
    [11]    [6.3000]    [3.3000]    [     6]
    [12]    [5.8000]    [2.7000]    [5.1000]
    [13]    [7.1000]    [     3]    [   NaN]
    [14]    [6.3000]    [2.9000]    [5.6000]
    [15]    [6.5000]    [     3]    [5.8000]

» mydata=cell2mat(mydata)

mydata =

    1.0000    5.1000       NaN    1.4000
    2.0000    4.9000    3.0000    1.4000
    3.0000    4.7000    3.2000    1.3000
    4.0000    4.6000    3.1000    1.5000
    5.0000    5.0000    3.6000       NaN
    6.0000    7.0000    3.2000    4.7000
    7.0000    6.4000    3.2000    4.5000
    8.0000    6.9000    3.1000    4.9000
    9.0000    5.5000    2.3000    4.0000
   10.0000       NaN    2.8000    4.6000
```

And now we run the k-NN with the **knnimpute( )** function:



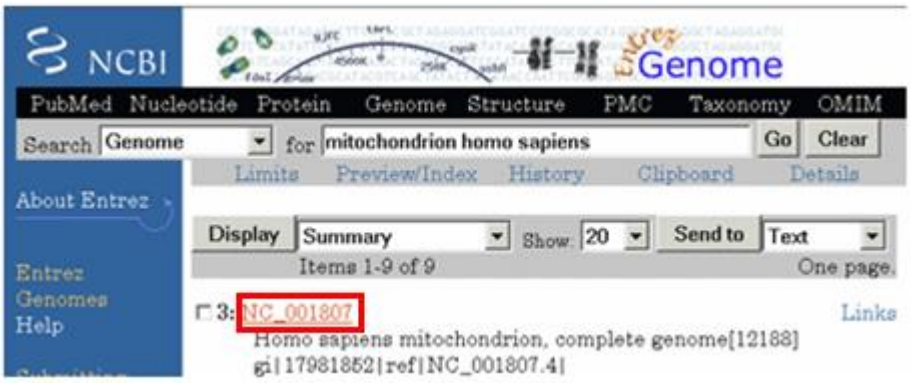
Access sequence databases

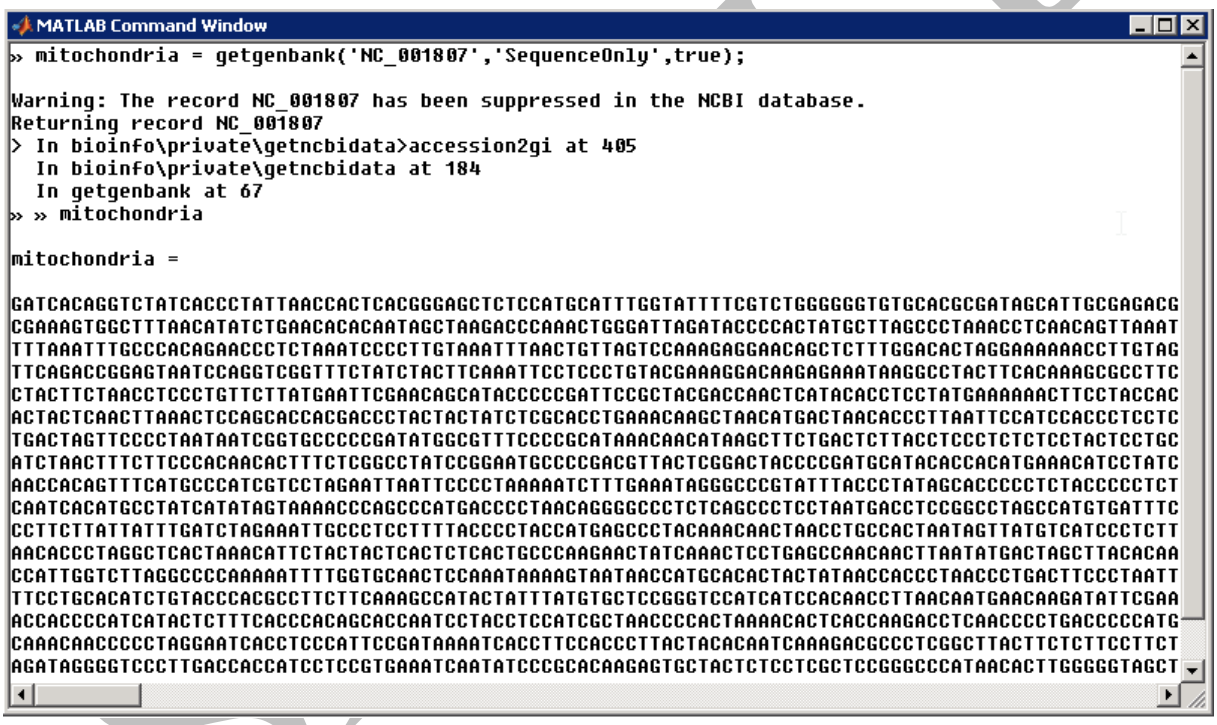
You can directly access public databases on the Web and copy sequence and gene expression information into MATLAB.

The sequence databases currently supported are GenBank (getgenbank), GenPept (getgenpept), European Molecular Biology Laboratory EMBL (getembl), Protein Sequence Database PIR-PSD (getpir), and Protein Data Bank PDB (getpdb). You can also access data

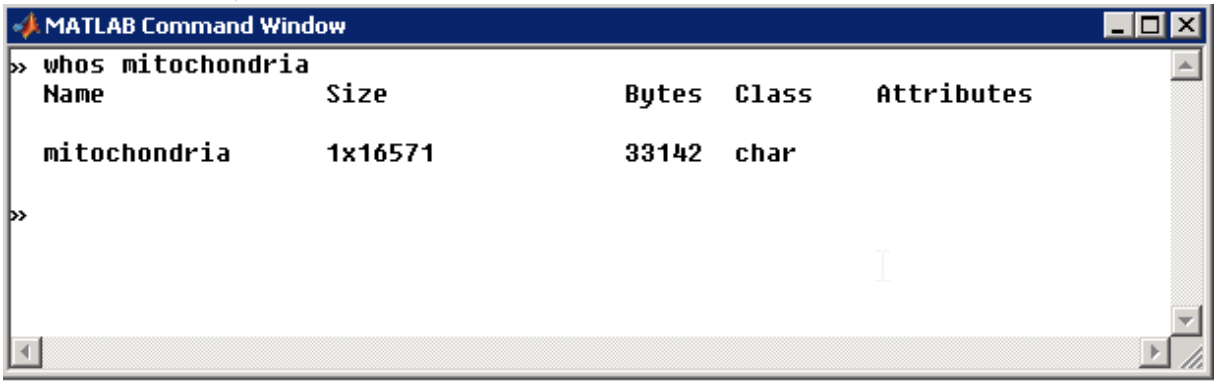
from the NCBI Gene Expression Omnibus (GEO) web site by using a single function (getgeodata).

Or you can ead data generated from gene sequencing instruments (scfread, joinseq, traceplot) and mass spectrometers (jcampread).





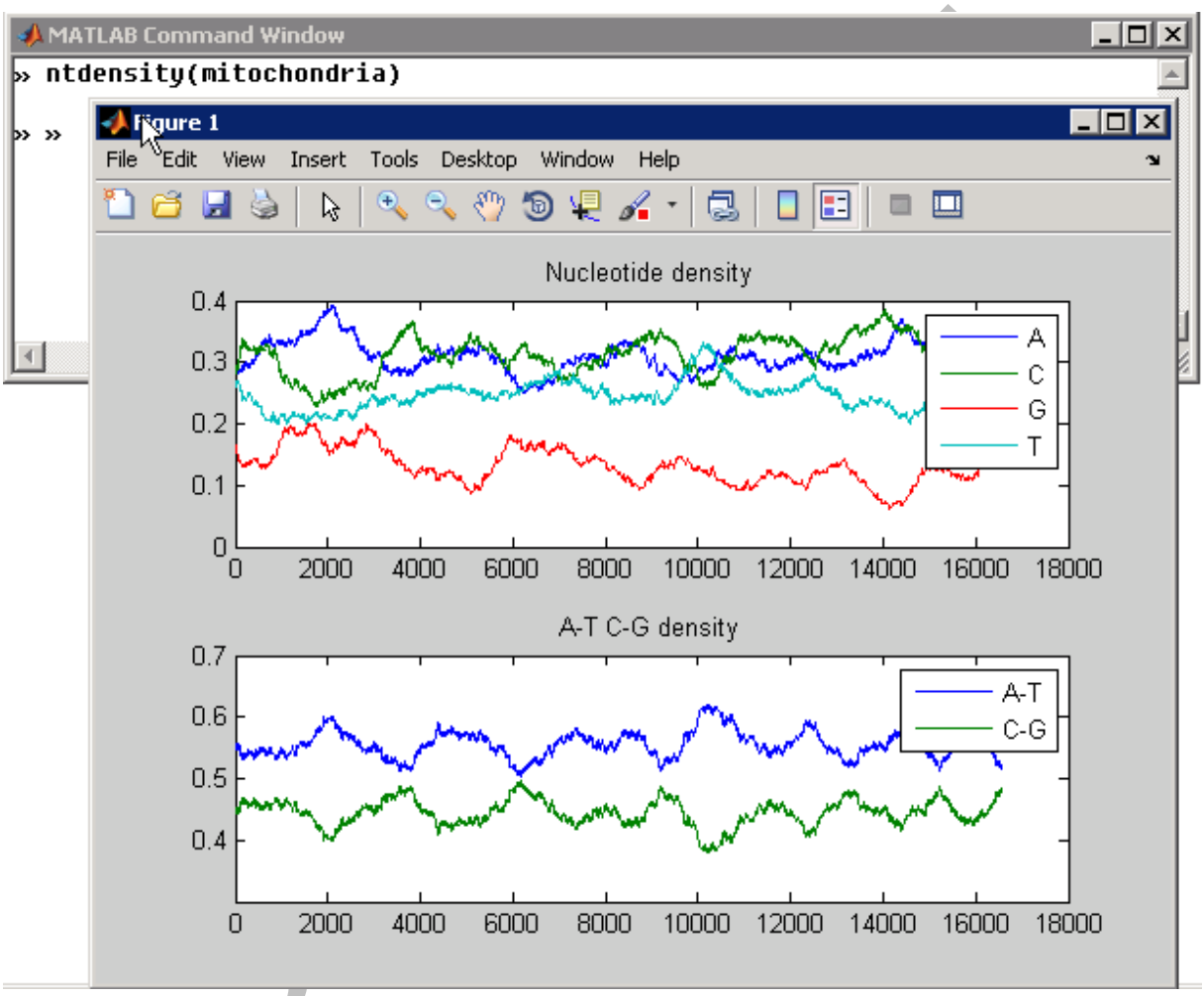
Note: If you don't have a Web connection, you can load the data from a MAT-file included with the Bioinformatics Toolbox, using the command **load** *mitochondria*



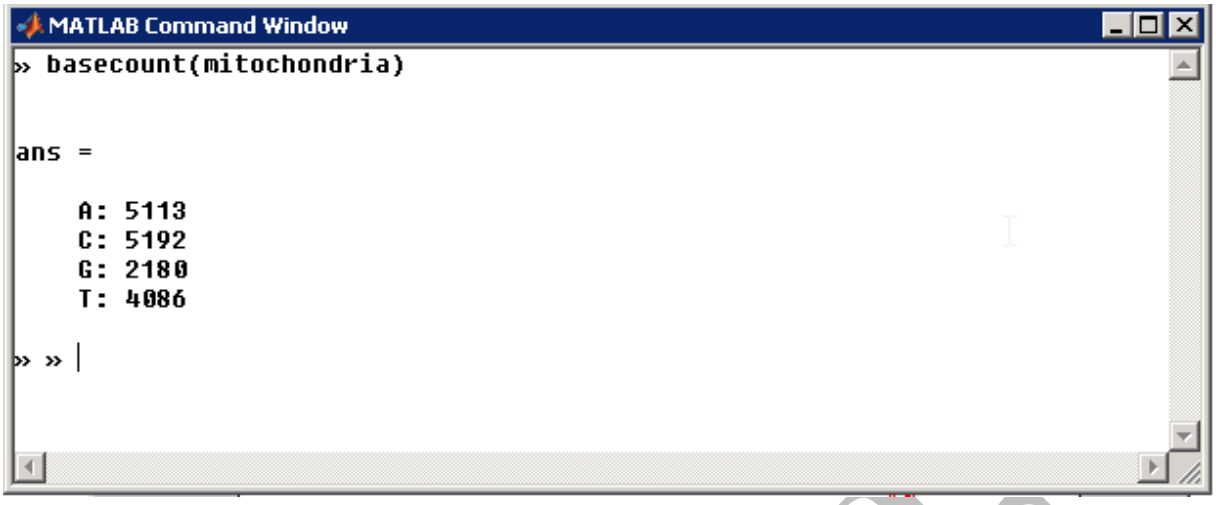
Sections of a DNA sequence with a high percent of A+T nucleotides usually indicates intergenic parts of the sequence, while low A+T and higher G+C nucleotide percentages indicate possible genes. Many times high CG dinucleotide content is located before a gene.

After you read a sequence into MATLAB, you can use the sequence statistics functions to determine if your sequence has the characteristics of a protein-coding region. This procedure uses the human mitochondrial genome as an example.

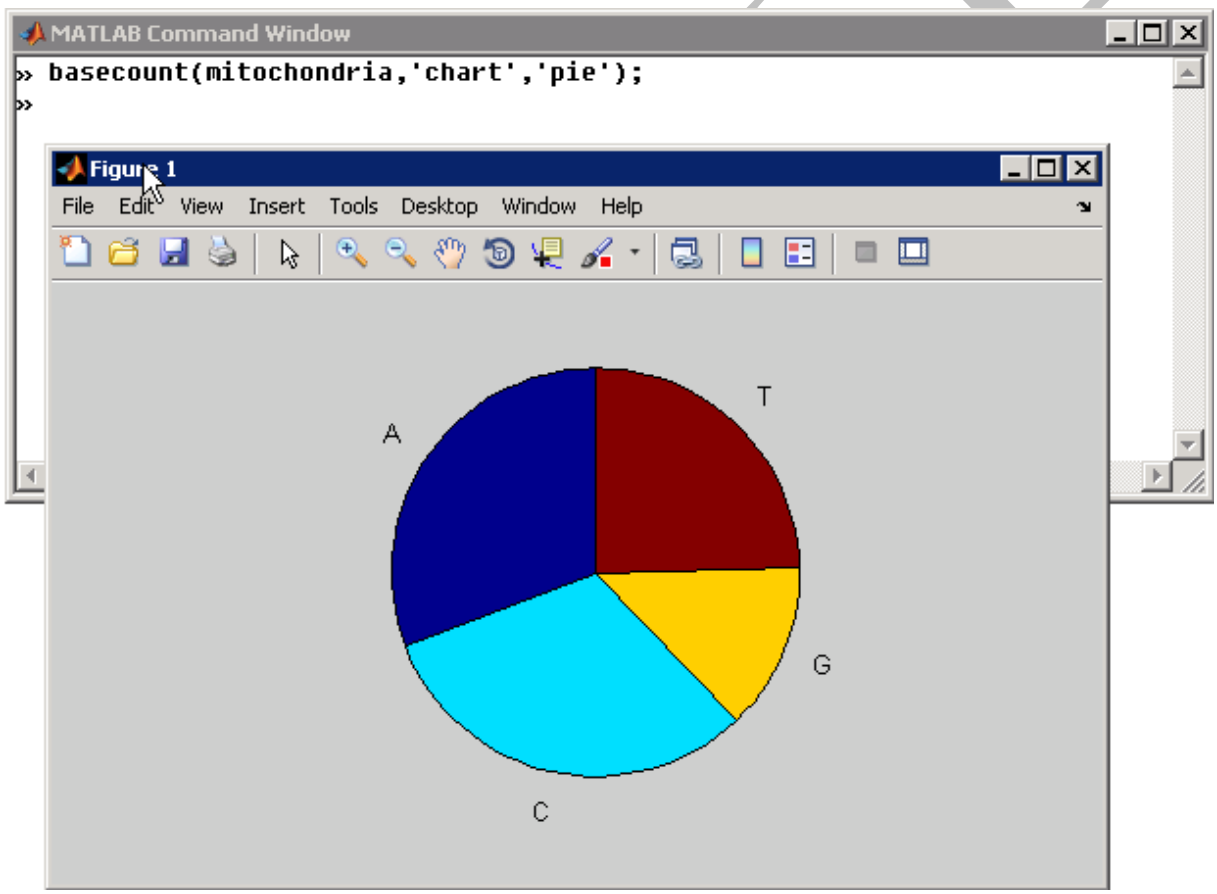
Plot monomer densities and combined monomer densities in a graph. In the MATLAB Command window, type:



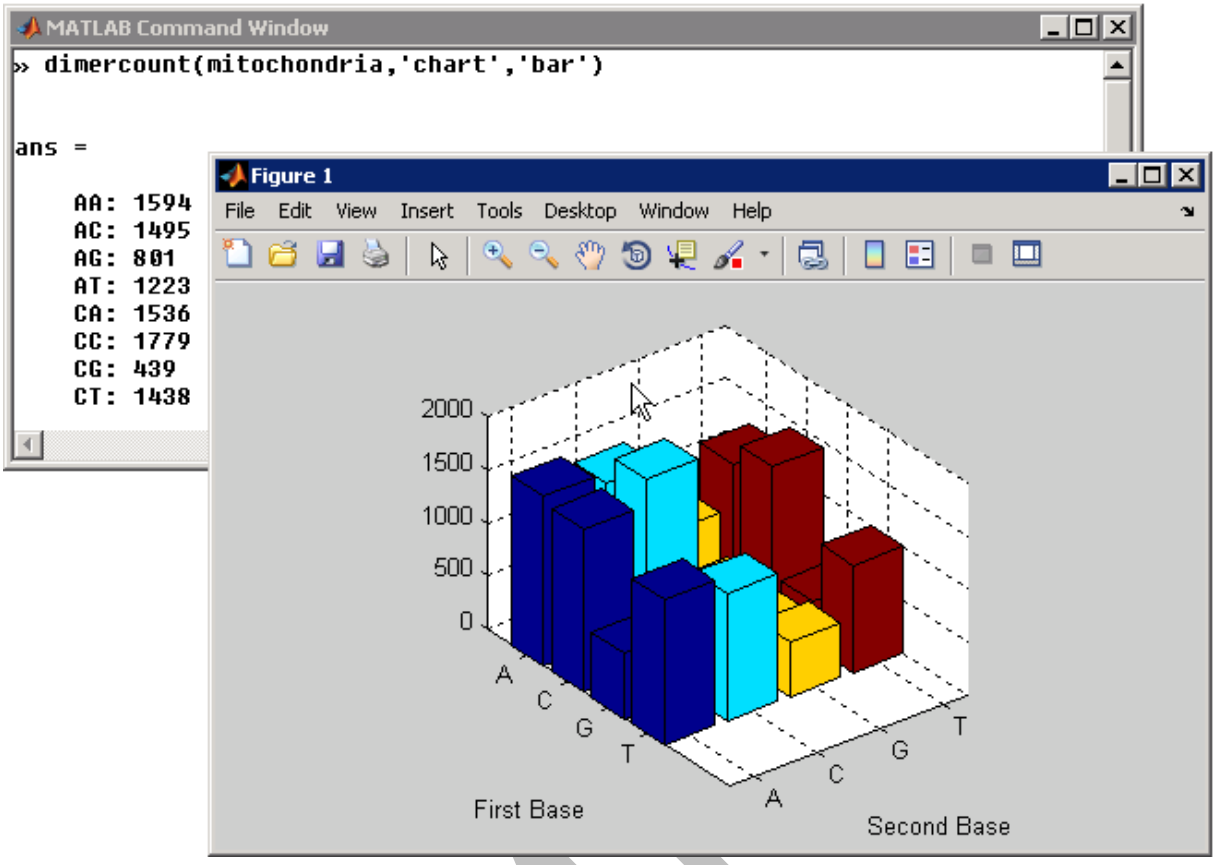
Count the nucleotides using the function **basecount( )**:



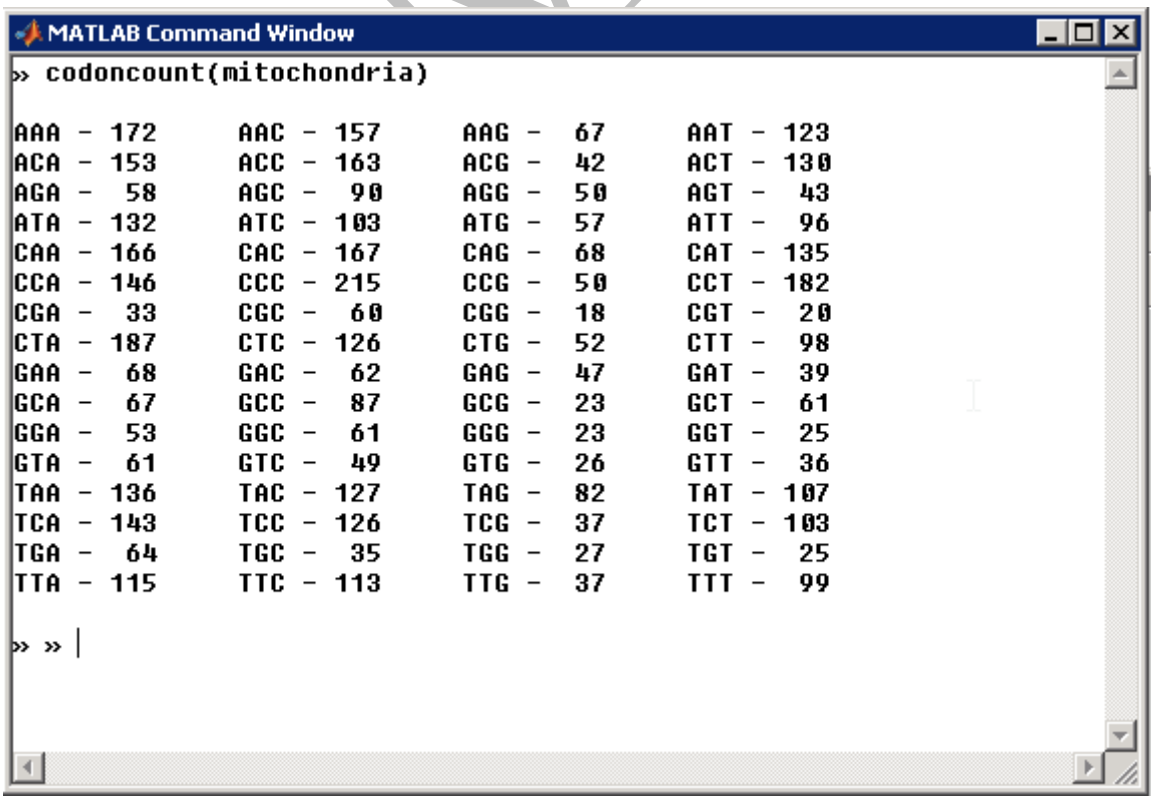
Use the function **basecount( )** with the chart option to visualize the nucleotide distribution:



Count the dimers in a sequence and display the information in a bar chart:

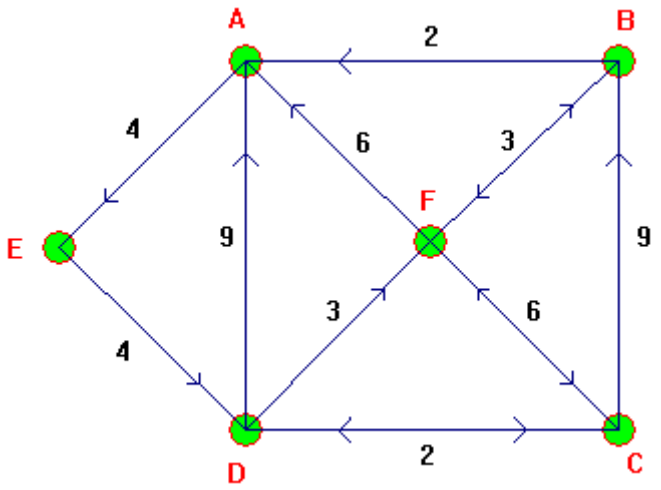


Trinucleotides (codon) code for an amino acid, and there are 64 possible codons in a nucleotide sequence. Knowing the percent of codons in your sequence can be helpful when you are comparing with tables for expected codon usage.



Shortest Path

We have study in the theoretical computing course the dijkra algorithm based on the following convex oriented chart:



It seems the bioinformatics toolbox can only manage one directional links or no directions at all. Thus we were obliged to simplify the example calculate by hand as you will see in the figure below.

The corresponding script using the functions **biograph( )**, **graphshortestpath( )** and **getedgesbynodeid( )**:

EDITOR PUBLISH VIEW

```
close all;clear all;clc
W = [2 6 3 9 6 3 2 9 4 4];
DG = sparse([2 6 6 3 3 4 3 4 1 5],[1 1 2 2 6 6 4 1 5 4],W);
h = view(biograph(DG,[],'ShowWeights','on'));
[dist,path] = graphshortestpath(DG,3,1)
set(h.Nodes(path),'Color',[1 0.4 0.4])
edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
set(edges,'LineColor',[1 0 0])
set(edges,'LineWidth',1.5)
```

And we get in the **Command Window**:

```
Command Window

dist =

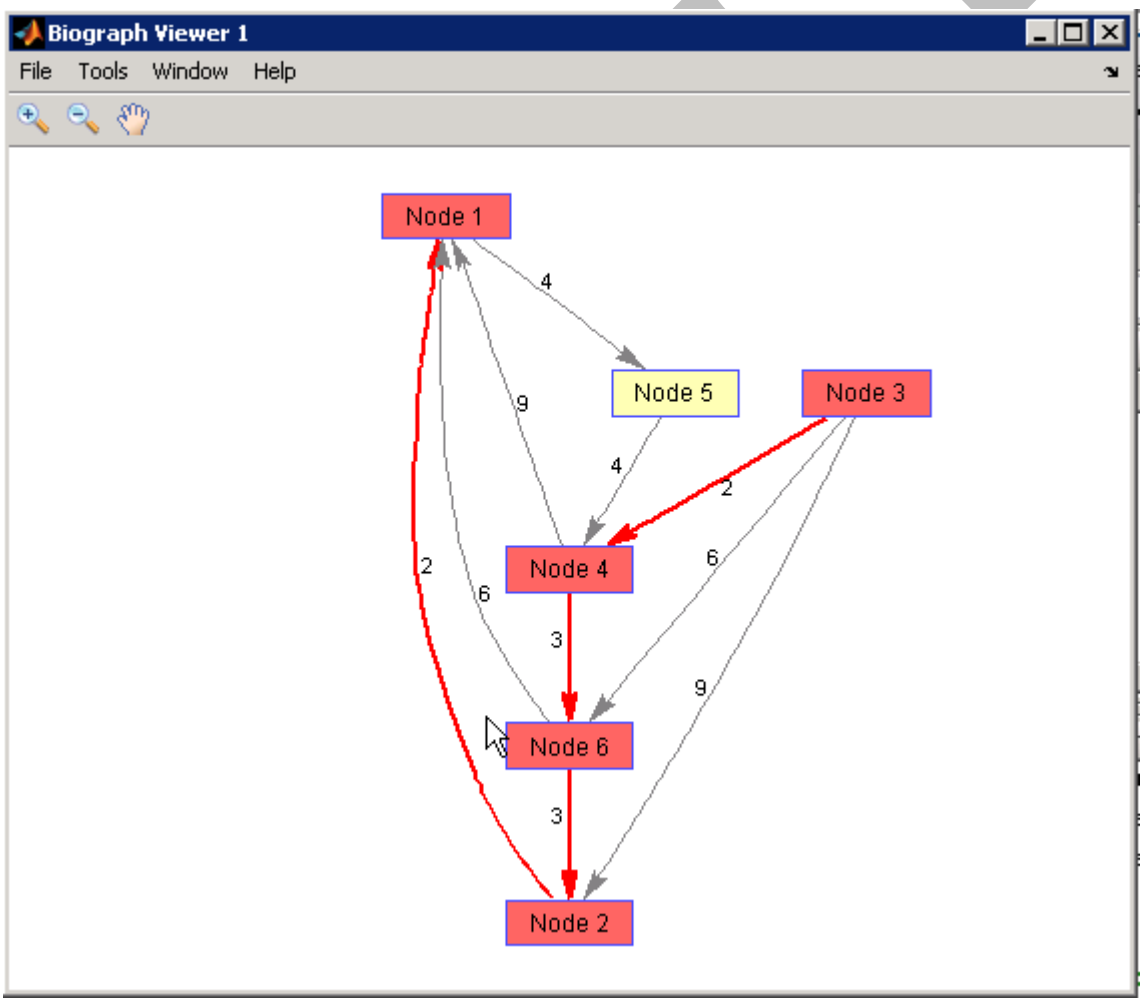
    10


path =

     3     4     6     2     1

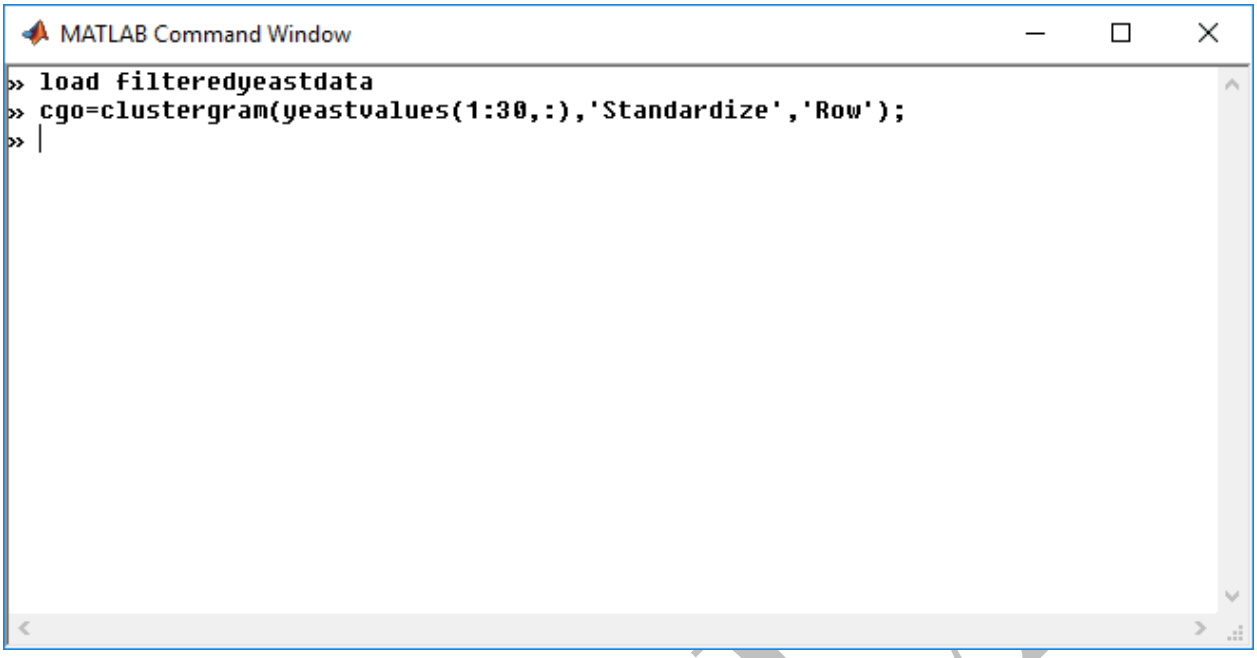
fx >>
```

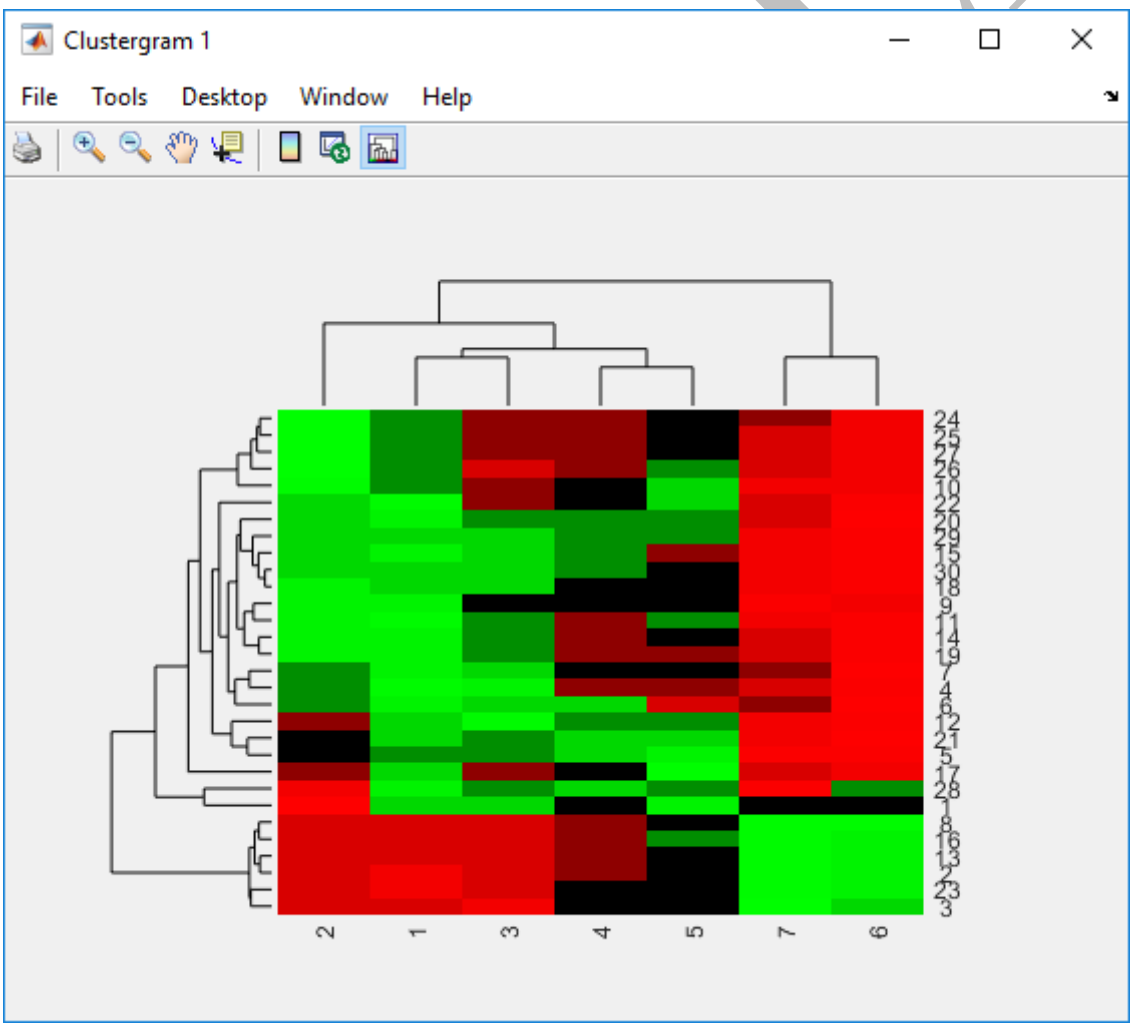
And the corresponding graph:



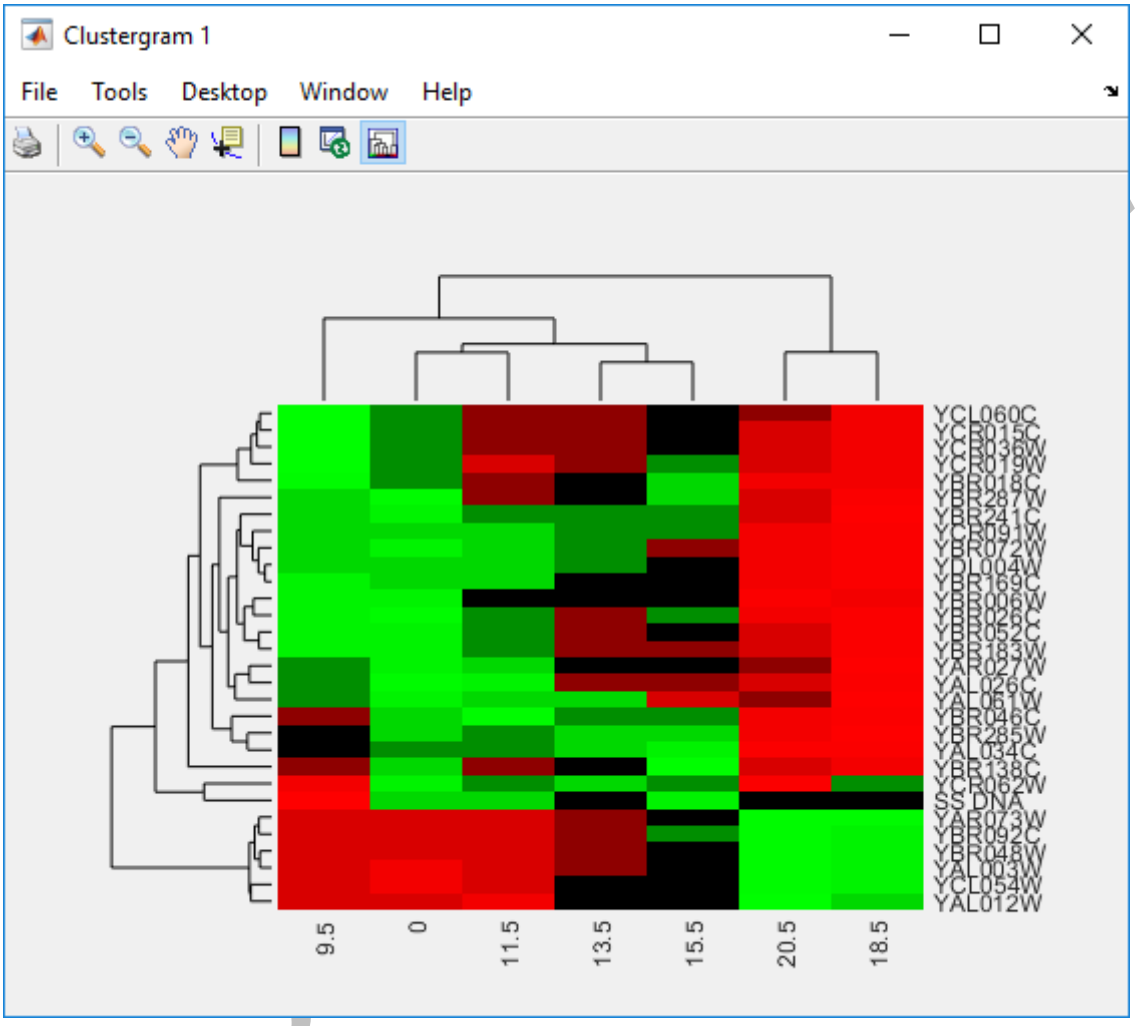
Clustermap

Then we use the function **clustergram( )**:

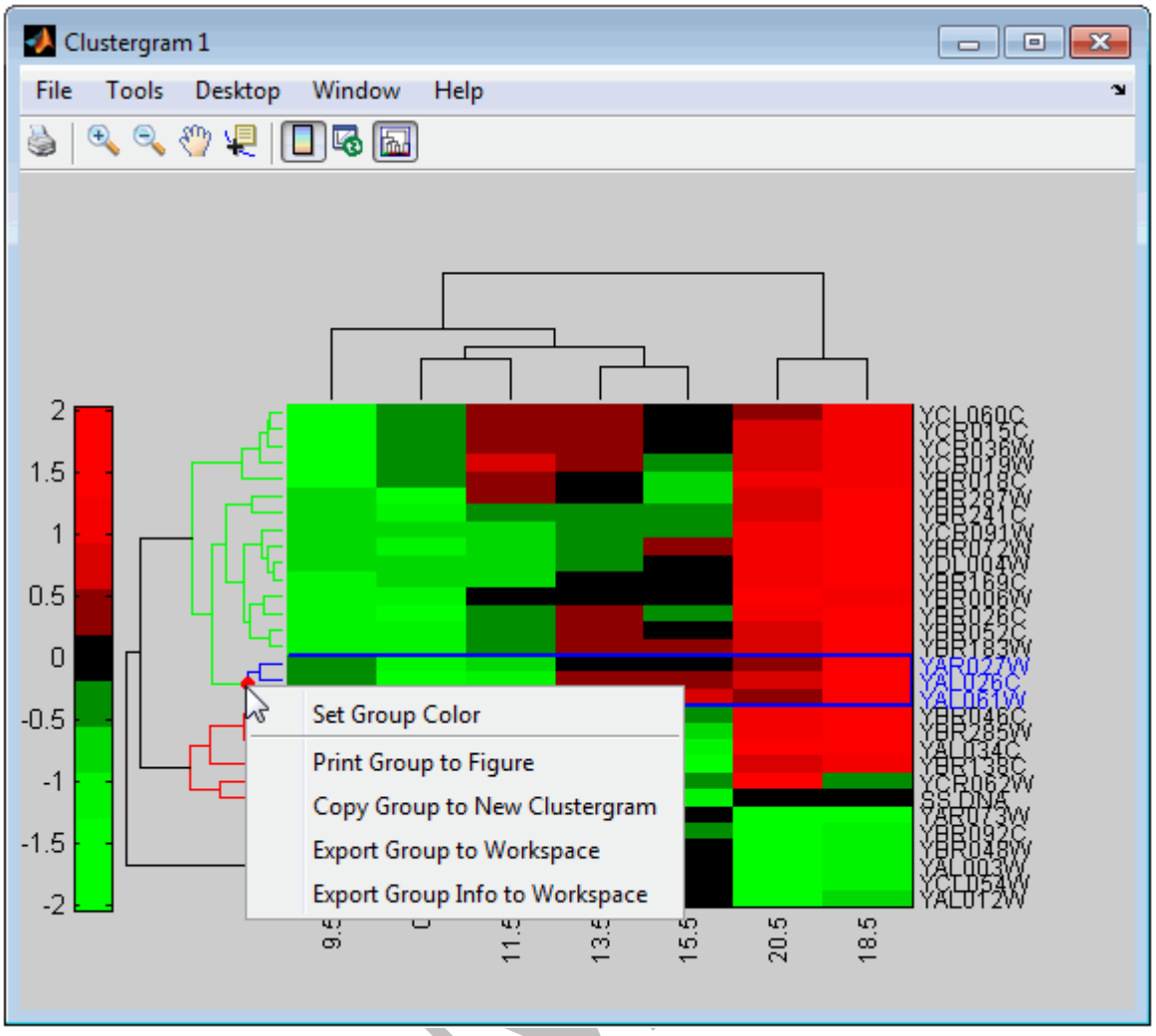

MATLAB Command Window
» load filteredyeastdata
» cgo=clustergram(yeastvalues(1:30,:),'Standardize','Row');
»

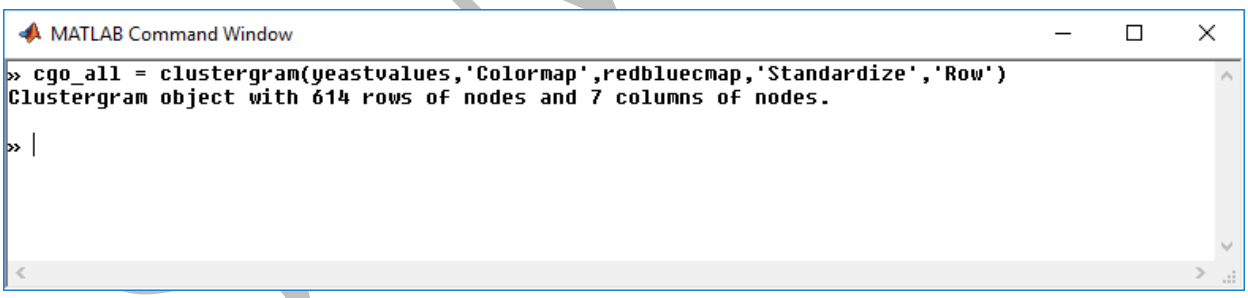

Clustergram 1
File Tools Desktop Window Help
24 25 27 26 10 22 20 29 15 30 18 9 11 14 19 7 4 6 12 21 5 17 28 1 8 16 13 2 23 3
2 1 3 4 5 7 6

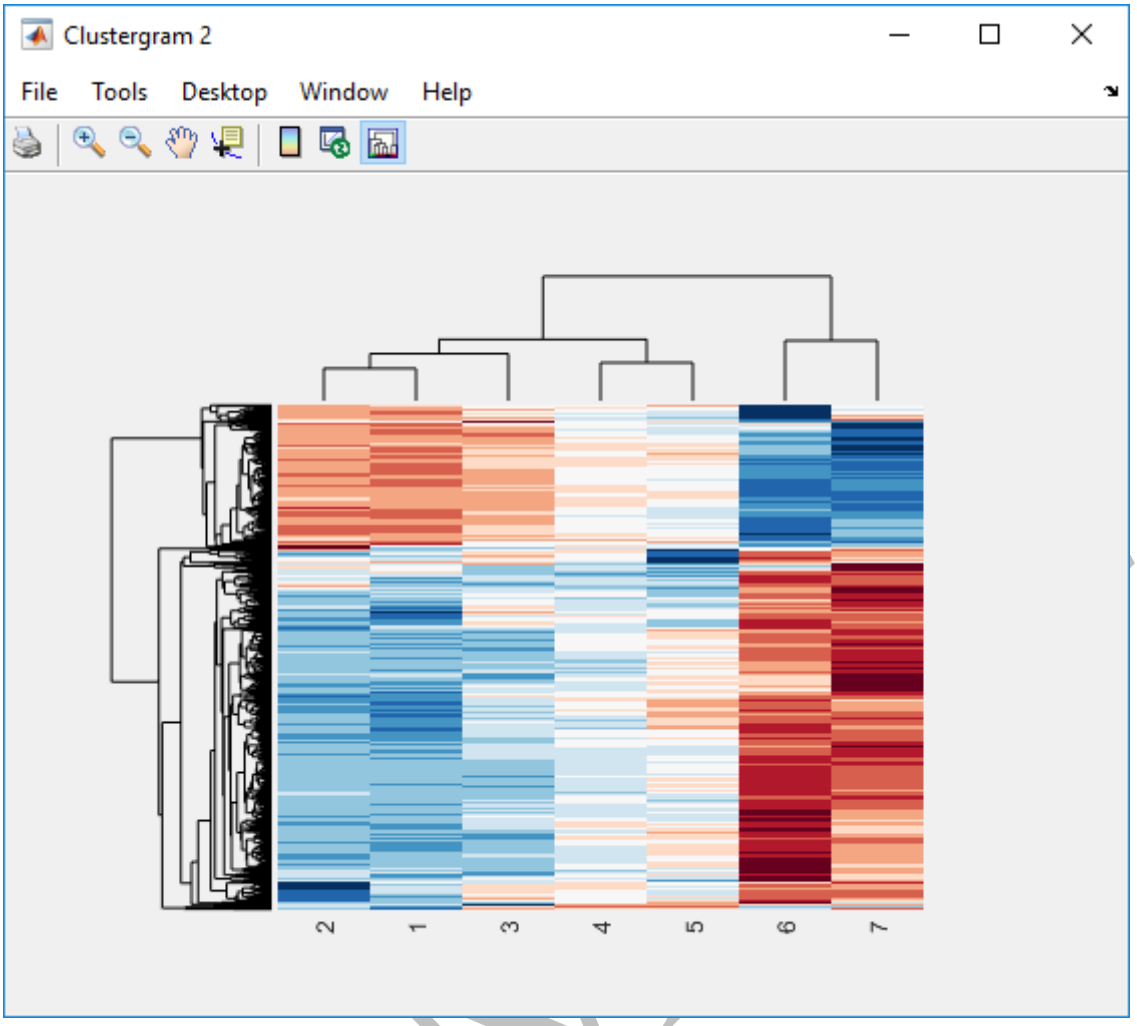
```
MATLAB Command Window
» set(cgo,'RowLabels',genes(1:30),'ColumnLabels',times)
»
```




```
MATLAB Command Window
» set(cgo,'Linkage','complete','Dendrogram',3)
»
```


Clustergram 1
File Tools Desktop Window Help
Set Group Color
Print Group to Figure
Copy Group to New Clustergram
Export Group to Workspace
Export Group Info to Workspace


MATLAB Command Window
» cgo_all = clustergram(yeastvalues,'Colormap',redbluecmap,'Standardize','Row')
Clustergram object with 614 rows of nodes and 7 columns of nodes.
»



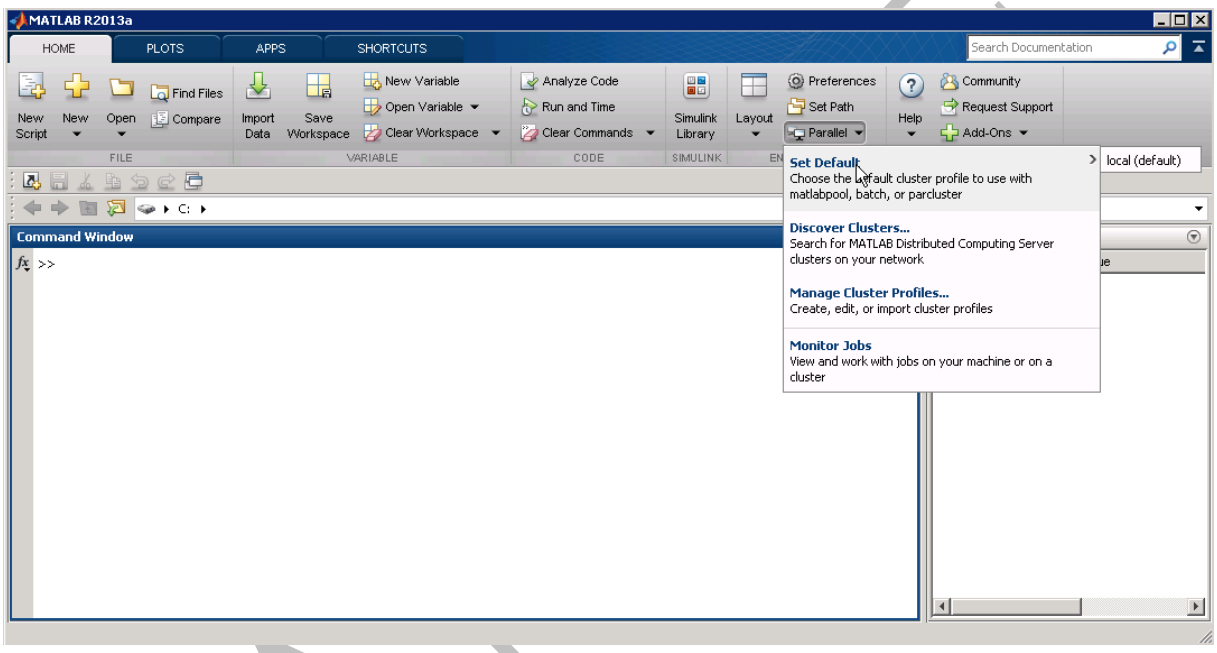
SAM

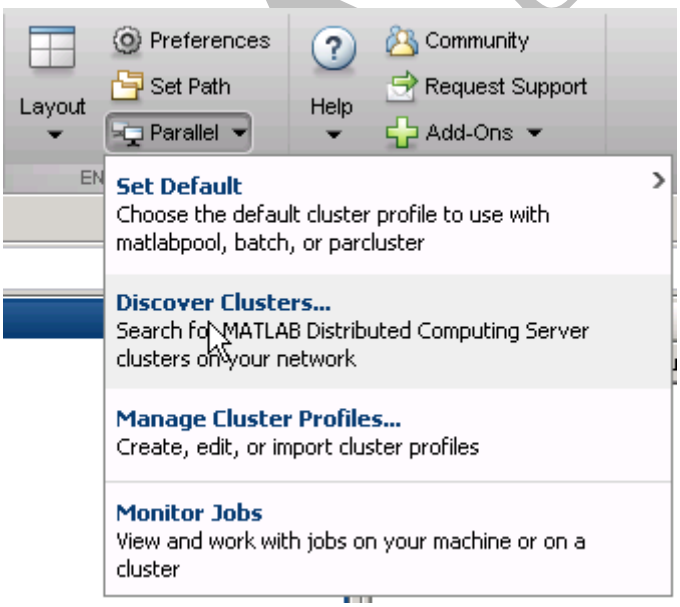
Parallel Computing (with Parallel Computing Toolbox/PCT)

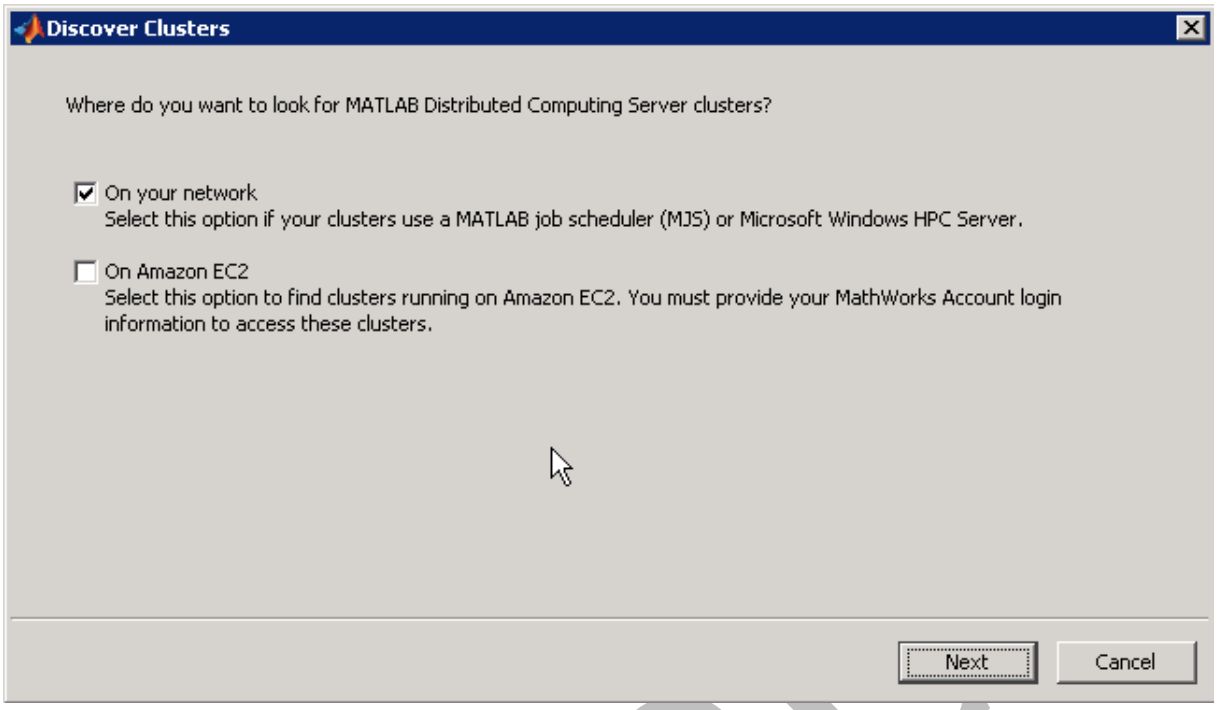
http://www.MathWorks.ch/ch/help/distcomp/release-notes.html

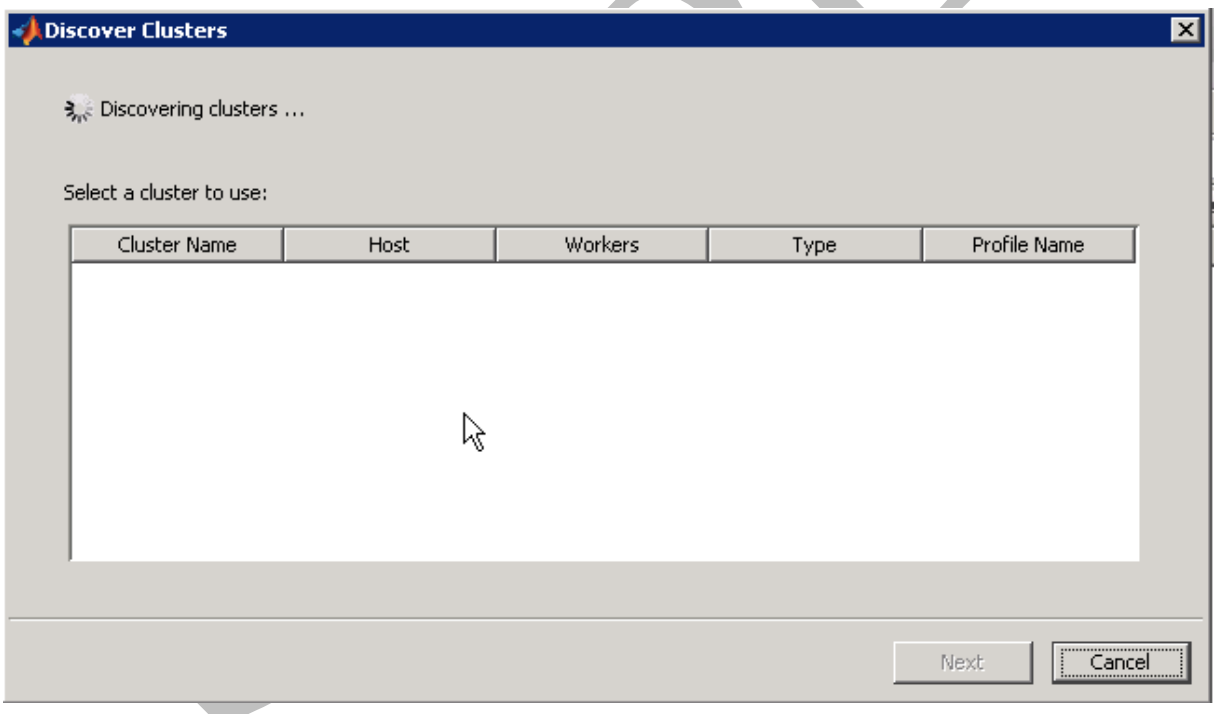
The PCT implementation supports since MATLAB 2014a up to 512 parallel workers on a single node. This is a big leap forward, given that MATLAB 2013b only supported 16 parallel workers!

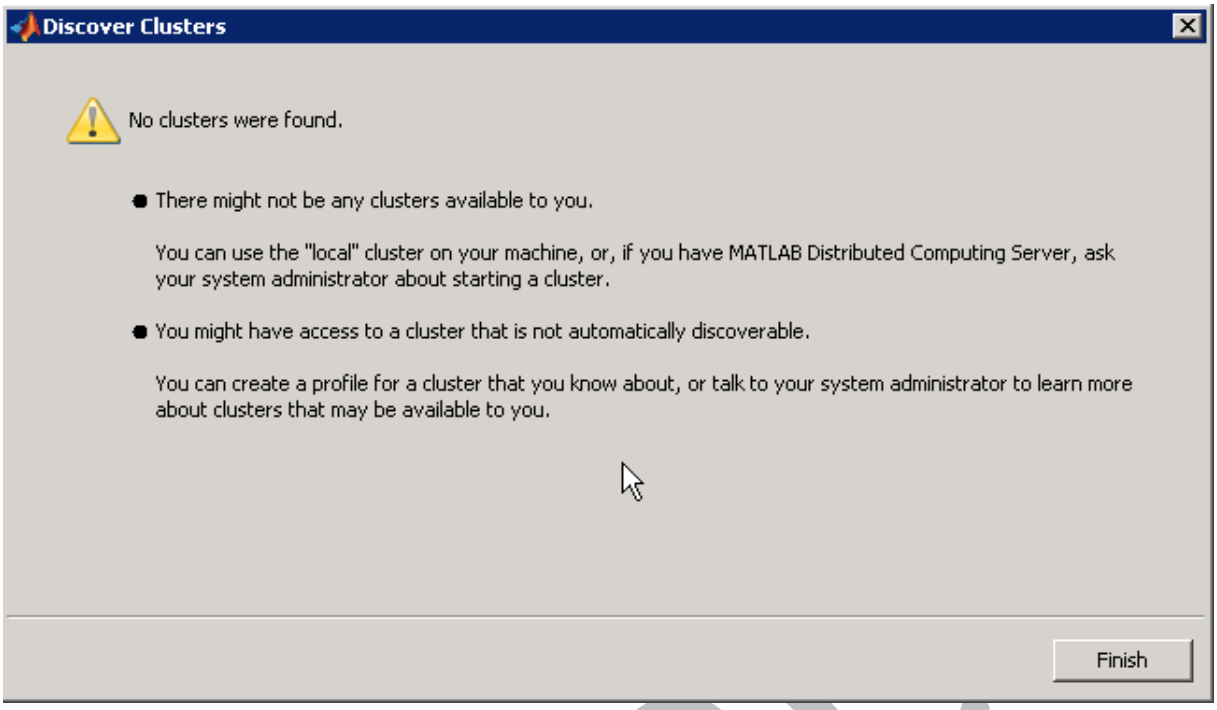
Clusters computing

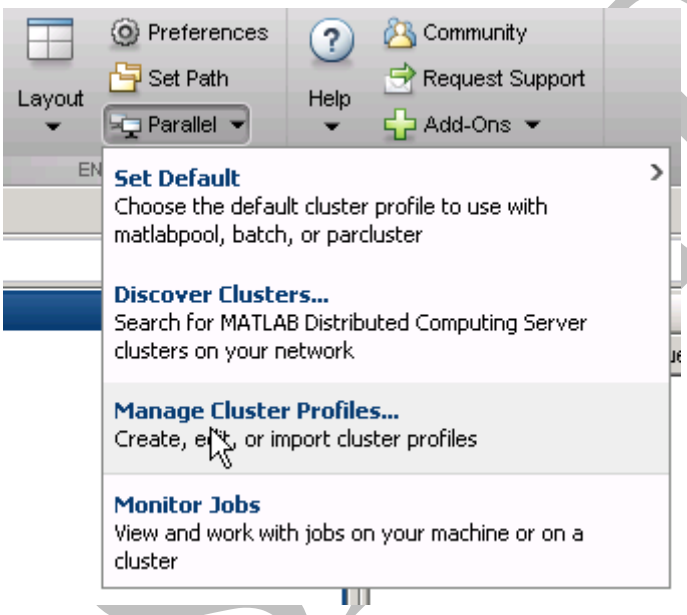


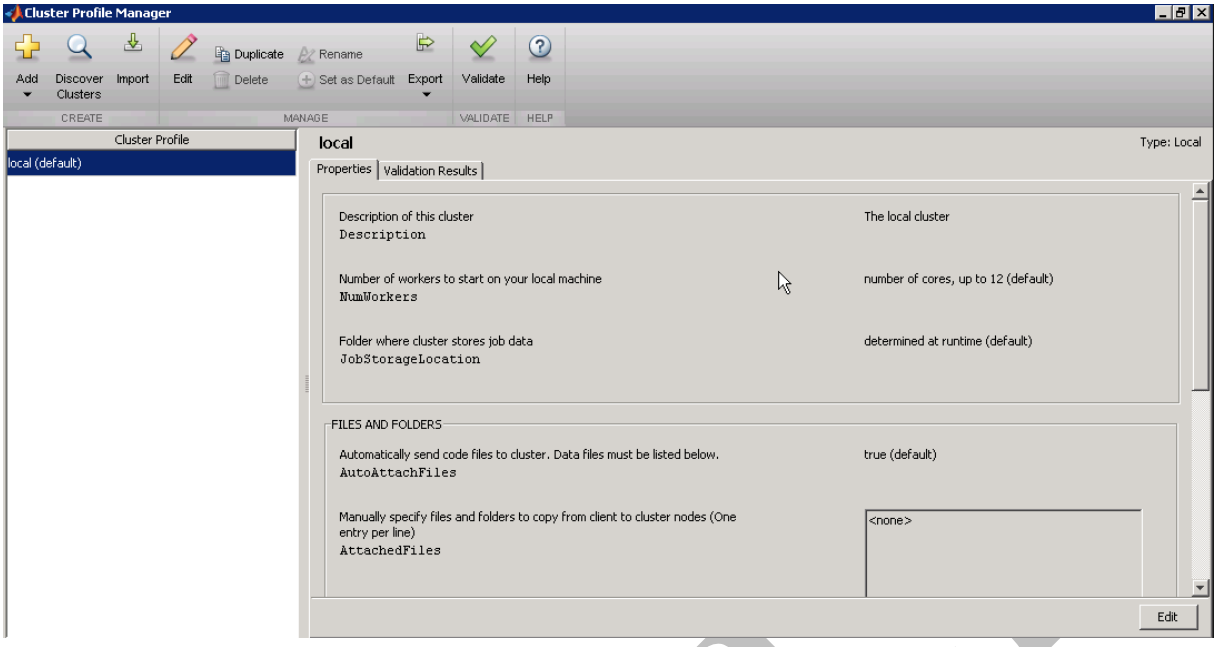


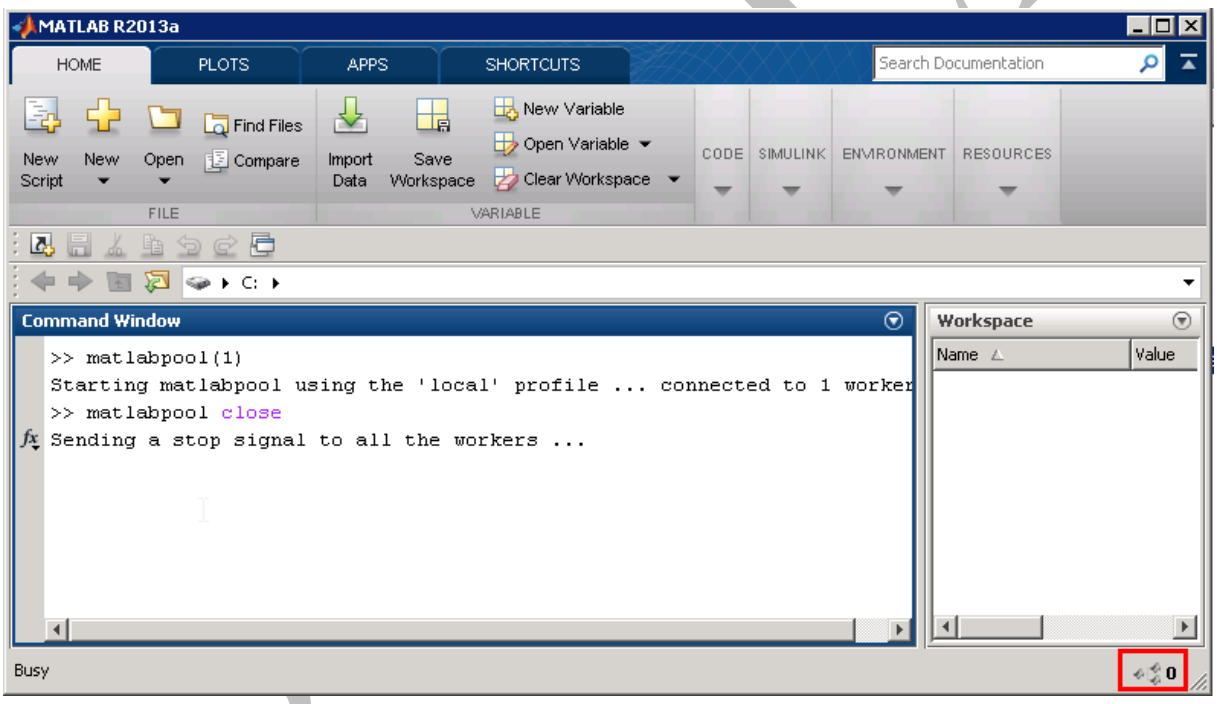

Discover Clusters
Where do you want to look for MATLAB Distributed Computing Server clusters?
On your network
Select this option if your clusters use a MATLAB job scheduler (MJS) or Microsoft Windows HPC Server.
On Amazon EC2
Select this option to find clusters running on Amazon EC2. You must provide your MathWorks Account login information to access these clusters.
Next
Cancel

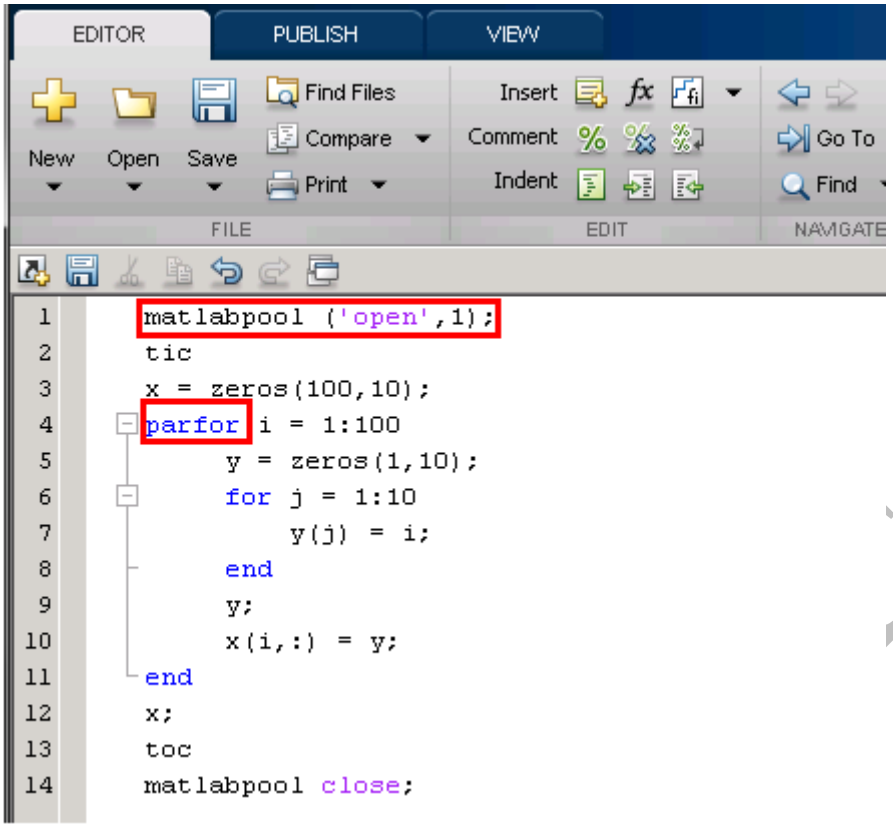

Discover Clusters
Discovering clusters ...
Select a cluster to use:
Cluster Name
Host
Workers
Type
Profile Name
Next
Cancel


Discover Clusters
No clusters were found.
There might not be any clusters available to you.
You can use the "local" cluster on your machine, or, if you have MATLAB Distributed Computing Server, ask your system administrator about starting a cluster.
You might have access to a cluster that is not automatically discoverable.
You can create a profile for a cluster that you know about, or talk to your system administrator to learn more about clusters that may be available to you.
Finish


Layout
Preferences
Set Path
Parallel
Help
Community
Request Support
Add-Ons
Set Default
Choose the default cluster profile to use with matlabpool, batch, or parcluster
Discover Clusters...
Search for MATLAB Distributed Computing Server clusters on your network
Manage Cluster Profiles...
Create, e
, or import cluster profiles
Monitor Jobs
View and work with jobs on your machine or on a cluster


Cluster Profile Manager
Add
Discover Clusters
Import
Edit
Duplicate
Delete
Rename
Set as Default
Export
Validate
Help
CREATE
MANAGE
VALIDATE
HELP
Cluster Profile
local (default)
local
Type: Local
Properties
Validation Results
Description of this cluster
Description
The local cluster
Number of workers to start on your local machine
NumWorkers
number of cores, up to 12 (default)
Folder where cluster stores job data
JobStorageLocation
determined at runtime (default)
FILES AND FOLDERS
Automatically send code files to cluster. Data files must be listed below.
AutoAttachFiles
true (default)
Manually specify files and folders to copy from client to cluster nodes (One entry per line)
AttachedFiles
<none>
Edit


MATLAB R2013a
HOME
PLOTS
APPS
SHORTCUTS
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
CODE
SIMULINK
ENVIRONMENT
RESOURCES
FILE
VARIABLE
C:
Command Window
>> matlabpool(1)
Starting matlabpool using the 'local' profile ... connected to 1 worker
>> matlabpool close
Sending a stop signal to all the workers ...
Workspace
Name
Value
Busy
0



Here are some tips:

- when using **parfor**, all the executions of the loop need to be independent from each other. This is such that the **parfor** loop doesn't even executes consecutively (i = 1, 2, 3, 4, …), but quite randomly (i = 3, 4, 59, 37, …) decided by MATLAB. This way, you cannot have any variable that can be modified by all the different processors at the same time, unless it has the proper dimensional index (x(i,j,k)).

- the **parfor** loop can only be done for one level, i.e., you cannot have a **parfor** loop inside another **parfor** loop. Sorry! However, the **parfor** loop doesn't need to be the outermost, it can be inside a regular **for** loop if needed.

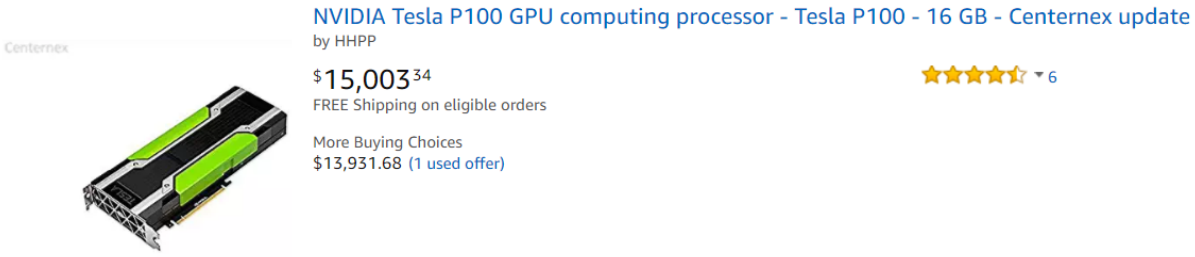
- You can have **for** loops inside the **parfor** loop, but you will need to pre-allocate all the variables that grow inside the **for** loops. Just pre-allocating immediately before the corresponding **for** loop will do. (This is usually a recommendation for speed in regular MATLAB runs, but it is a must in parallel mode.)

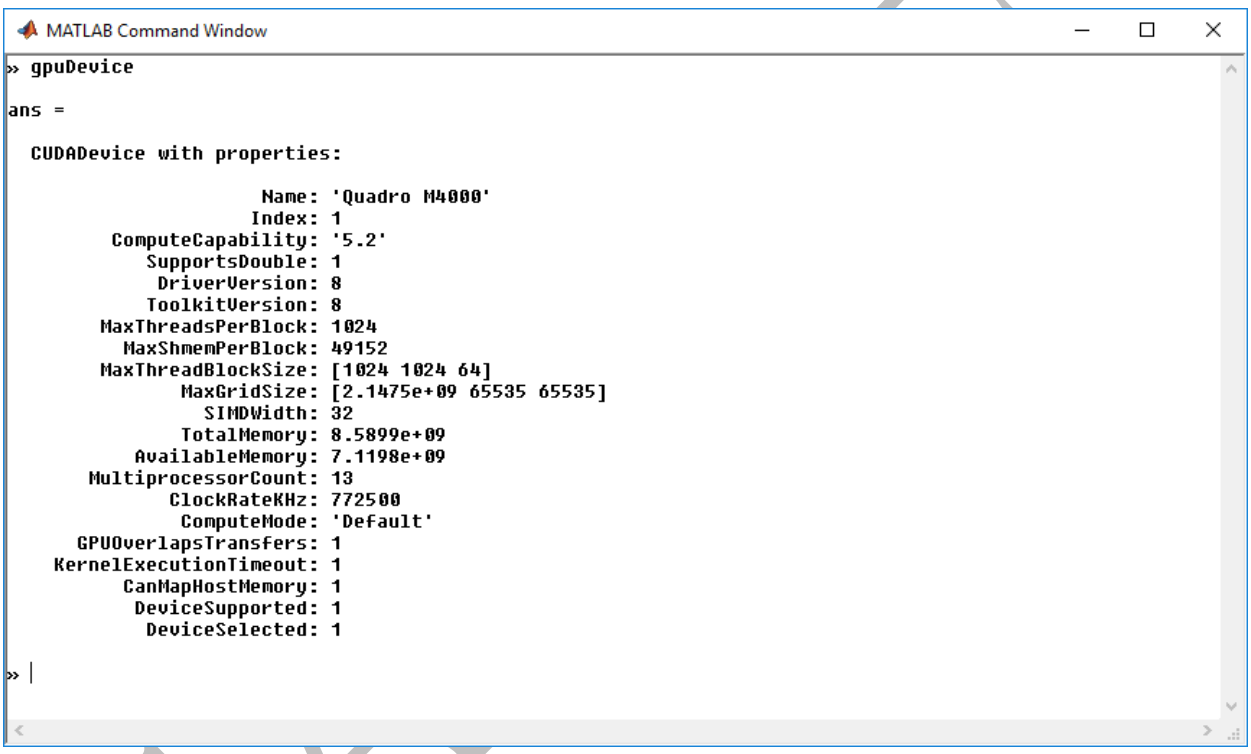
- You cannot have any instruction that calls to the workspace inside the **parfor** loop, such as delete variables (**clear**) or list variables (**who**).

- Be careful if you have figures inside the **parfor** loop. They should work, and you can even save them. But if you are displaying on screen, make sure you close the figures within the loop, so that they don't flood your desktop!

NVidia GPU computing

https://developer.nvidia.com/cuda-gpus





Editor - C:\Users\booth09-mgr2\Documents\MATLAB\GPU.m

GPU.m

```
clear all;close all;clc

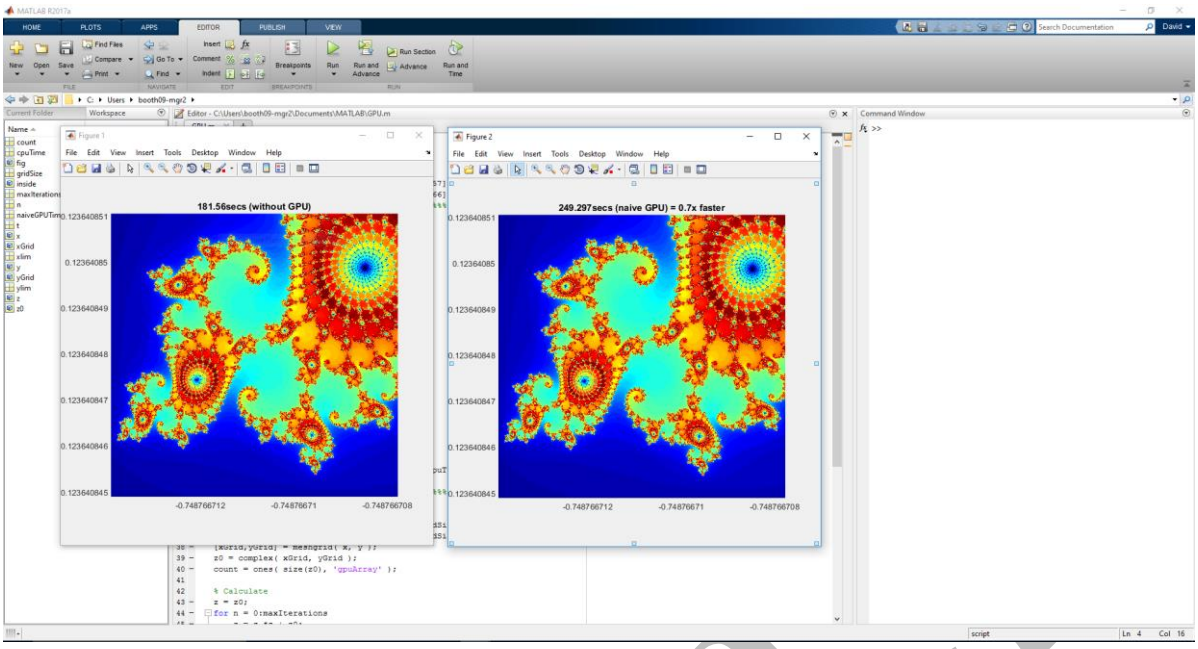
maxIterations = 500;
gridSize = 9000;
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862,  0.123640851045266];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t = tic();
x = linspace( xlim(1), xlim(2), gridSize );
y = linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
z0 = xGrid + 1i*yGrid;
count = ones( size(z0) );
% Calculate
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
count = log( count );
% Show
figure
hold on
cpuTime = toc( t );
fig = gcf;
fig.Position = [200 200 600 600];
imagesc( x, y, count );
axis image
colormap( [jet();flipud( jet() );0 0 0] );
title( sprintf( '%1.2fsecs (without GPU)', cpuTime ) );
hold off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup
t = tic();
x = gpuArray.linspace( xlim(1), xlim(2), gridSize );
y = gpuArray.linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
z0 = complex( xGrid, yGrid );
count = ones( size(z0), 'gpuArray' );

% Calculate
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
```

```
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
count = log( count );

% Show
figure
fig = gcf;
fig.Position = [200 200 600 600];
hold on
count = gather( count ); % Fetch the data back from the GPU
naiveGPUTime = toc( t );
imagesc( x, y, count )
axis image
colormap( [jet();flipud( jet() );0 0 0] );
title( sprintf( '%1.3fsecs (naive GPU) = %1.1fx faster', ...
    naiveGPUTime, cpuTime/naiveGPUTime ) )
hold off
```

It is not always faster with the GPU and especially when the GPU reach its limits:



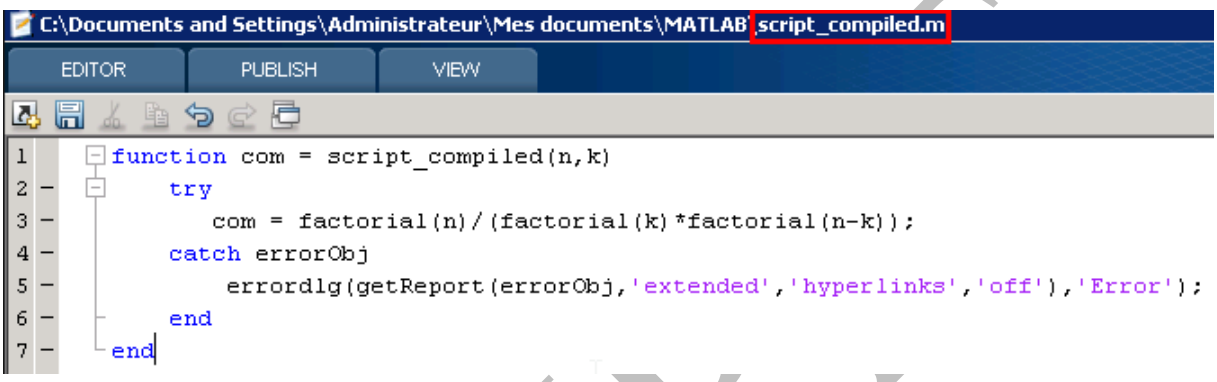
SAMPLE

Compiler Toolbox (C#, C++, Compiler Toolbox)

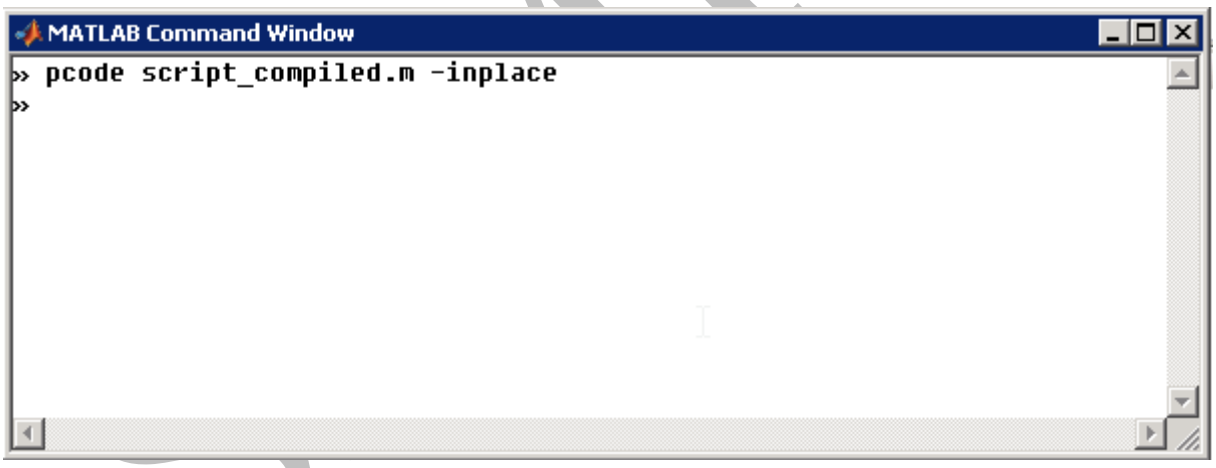
*Create a *.p (protected script) file (aka "pcode")*

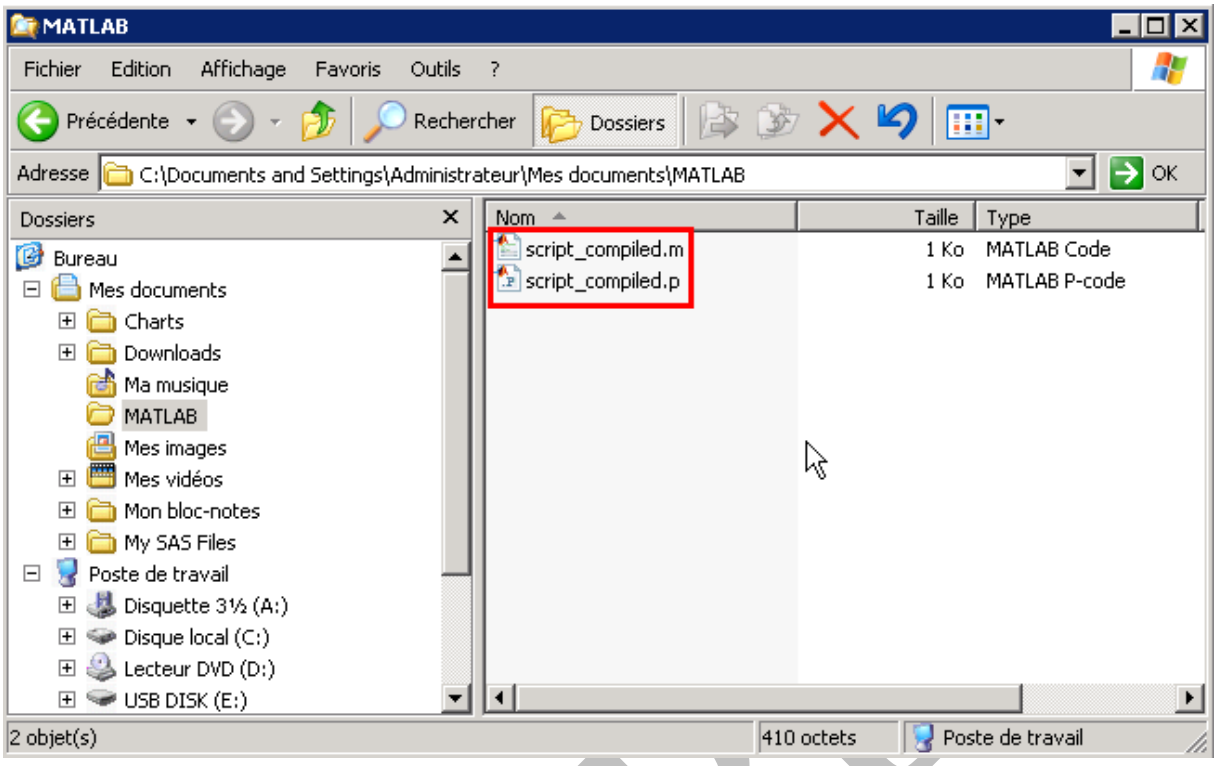
For this part you do not need any compiler but in my point of view this is the good place to speak about this feature.

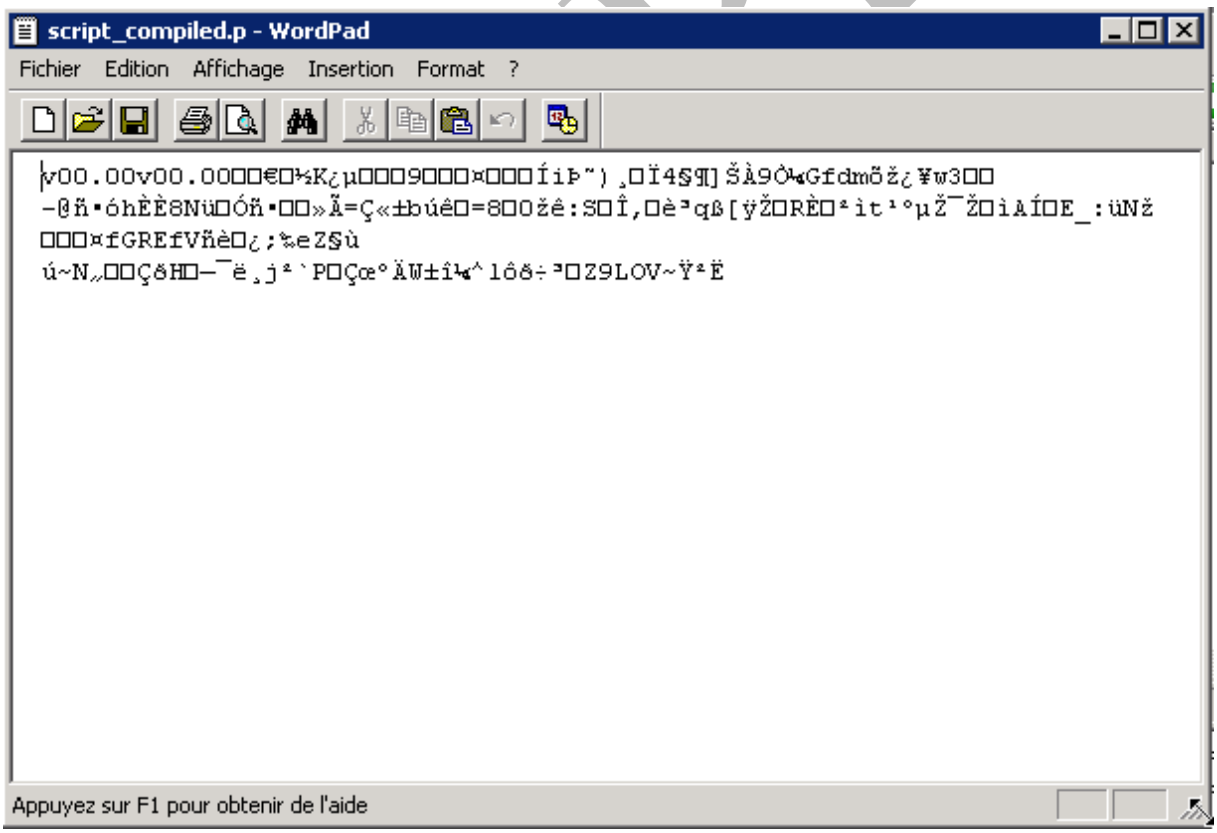
Consider the following script:



We want to make this an unreadable file (or difficult to read...) for other colleagues.





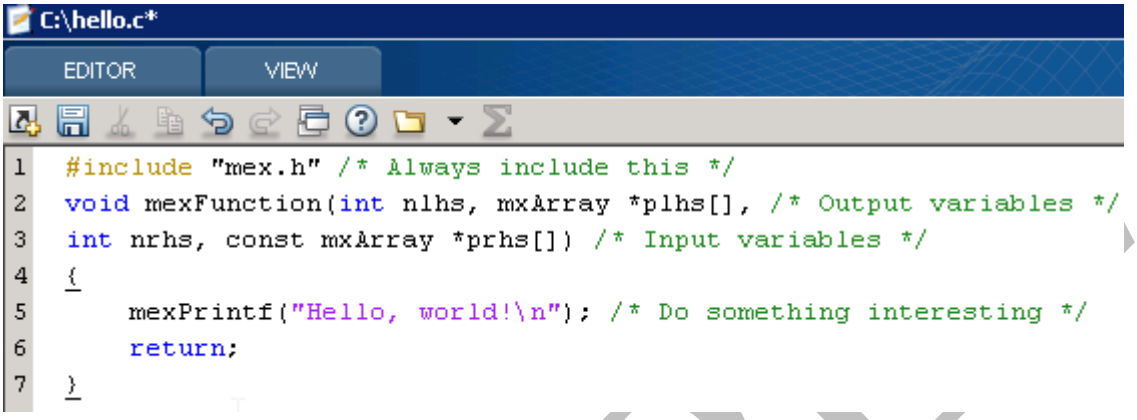


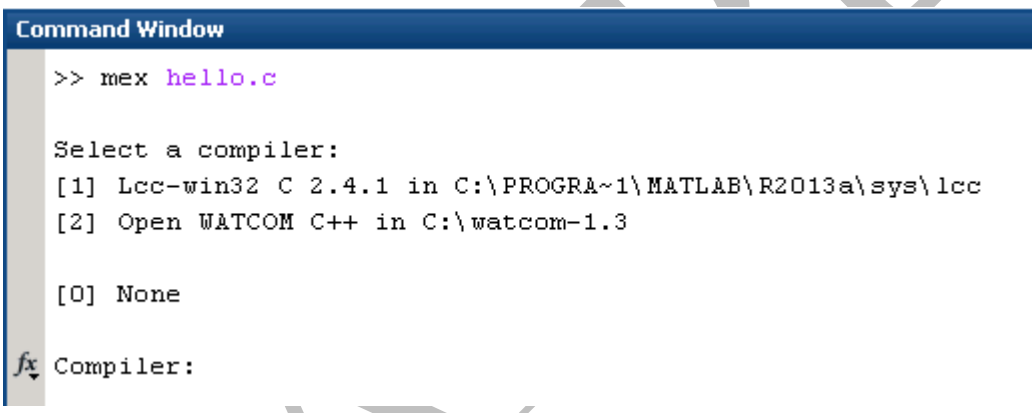
C MEX file

It is possible to compile C, C++, or Fortran code so that it is callable from MATLAB. This kind of program is called a MATLAB Executable (MEX) external interface function, or more briey a \MEX-function." MEX enables the high performance of C, C++, and Fortran while

working within the MATLAB environment. We will discuss C/MEX functions, which also applies directly to C++/MEX.Fortran/MEX is quite different and we do not discuss it here.

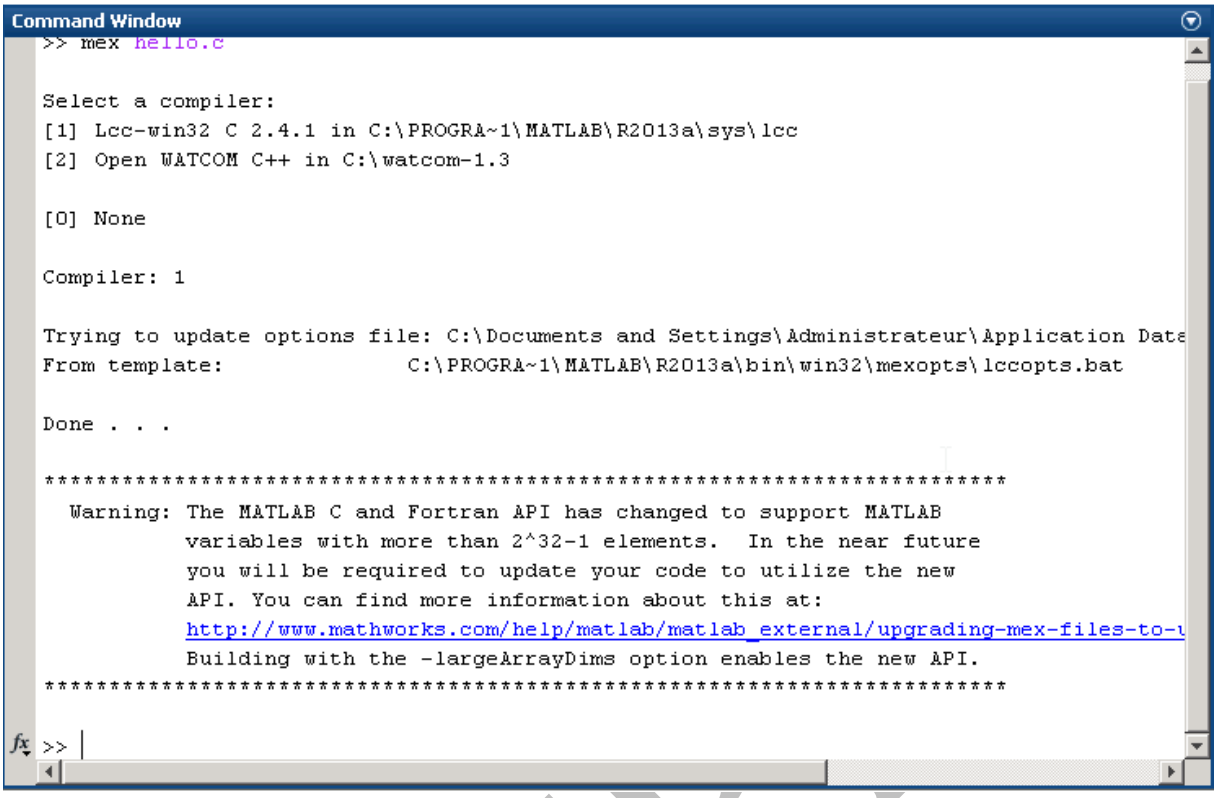
MATLAB's JIT interpreter in recent versions runs M-code so e_ciently that it is often times di_cult to do much better with C. Before turning to MEX in an application, optimize your M-code (see my other article, \Writing Fast MATLAB Code"). MEX-functions are best suited to substitute one or two bottleneck M-functions in an application. If you replace all functions in an application with MEX, you might as well port the application entirely to C.

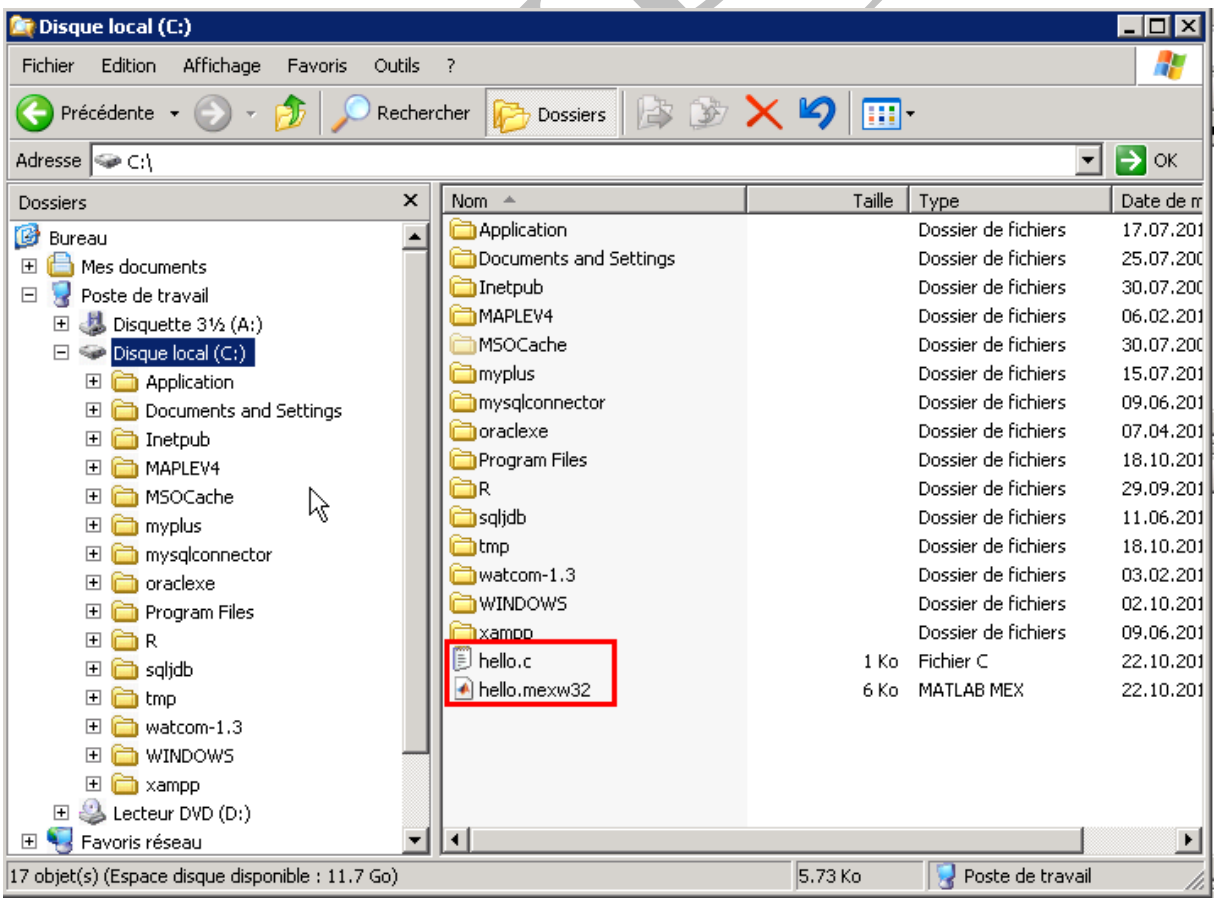


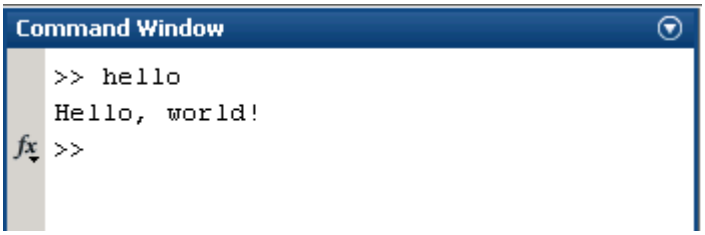


If successful, this command produces a compiled _le called hello.mexa64 (or similar, depending on platform). Compiling requires that you have a C compiler and that MATLAB is con_gured to use it. MATLAB will autodetect most popular compilers, including Microsoft Visual C/C++ and GCC. As a fallback, some distributions of MATLAB come with the Lcc C compiler. Run mex -setup to change the selected compiler and build settings.

Note that compiled MEX _les might not be compatible between di_erent platforms or di_erent versions of MATLAB. They should be compiled for each platform/version combination that you need. It is possible to compile a MEX _le for a target platform other than the host's using the -<arch> option, for example, mex -win32 hello.c.

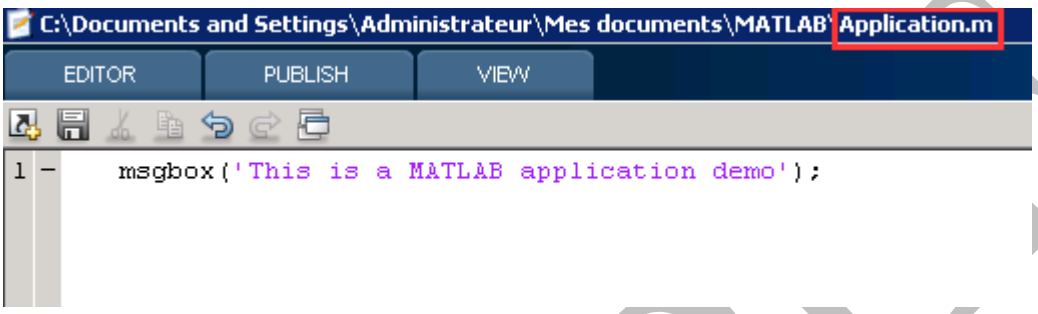

Command Window
>> mex hello.c
Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2013a\sys\lcc
[2] Open WATCOM C++ in C:\watcom-1.3
[0] None
Compiler: 1
Trying to update options file: C:\Documents and Settings\Administrateur\Application Date
From template: C:\PROGRA~1\MATLAB\R2013a\bin\win32\mexopts\lccopts.bat
Done . . .
Warning: The MATLAB C and Fortran API has changed to support MATLAB
variables with more than 2^32-1 elements. In the near future
you will be required to update your code to utilize the new
API. You can find more information about this at:
http://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-
Building with the -largeArrayDims option enables the new API.
>>

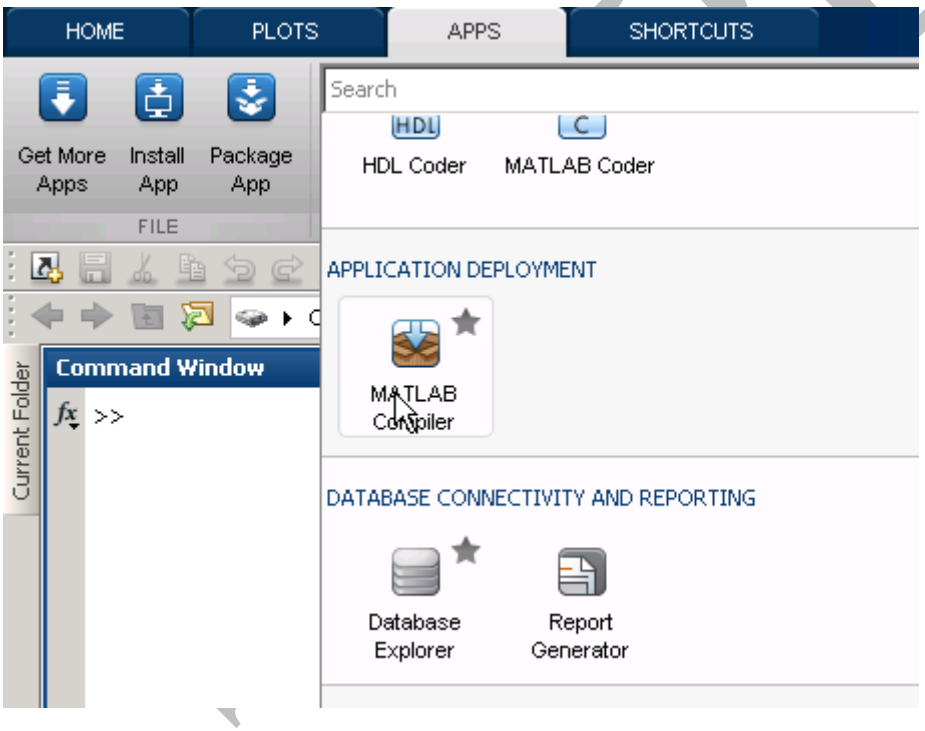

Disque local (C:)
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Dossiers
Adresse C:\
OK
Dossiers
Bureau
Mes documents
Poste de travail
Disquette 3½ (A:)
Disque local (C:)
Application
Documents and Settings
Inetpub
MAPLEV4
MSOCache
myplus
mysqlconnector
oraclexe
Program Files
R
sqljdb
tmp
watcom-1.3
WINDOWS
xampp
Lecteur DVD (D:)
Favoris réseau
Nom Taille Type Date de m
Application Dossier de fichiers 17.07.201
Documents and Settings Dossier de fichiers 25.07.200
Inetpub Dossier de fichiers 30.07.200
MAPLEV4 Dossier de fichiers 06.02.201
MSOCache Dossier de fichiers 30.07.200
myplus Dossier de fichiers 15.07.201
mysqlconnector Dossier de fichiers 09.06.201
oraclexe Dossier de fichiers 07.04.201
Program Files Dossier de fichiers 18.10.201
R Dossier de fichiers 29.09.201
sqljdb Dossier de fichiers 11.06.201
tmp Dossier de fichiers 18.10.201
watcom-1.3 Dossier de fichiers 03.02.201
WINDOWS Dossier de fichiers 02.10.201
xampp Dossier de fichiers 09.06.201
hello.c 1 Ko Fichier C 22.10.201
hello.mexw32 6 Ko MATLAB MEX 22.10.201
17 objet(s) (Espace disque disponible : 11.7 Go)
5.73 Ko
Poste de travail

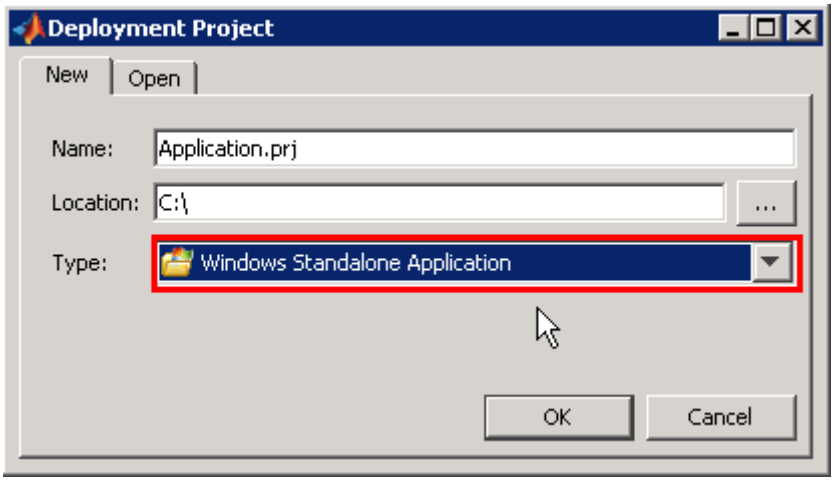


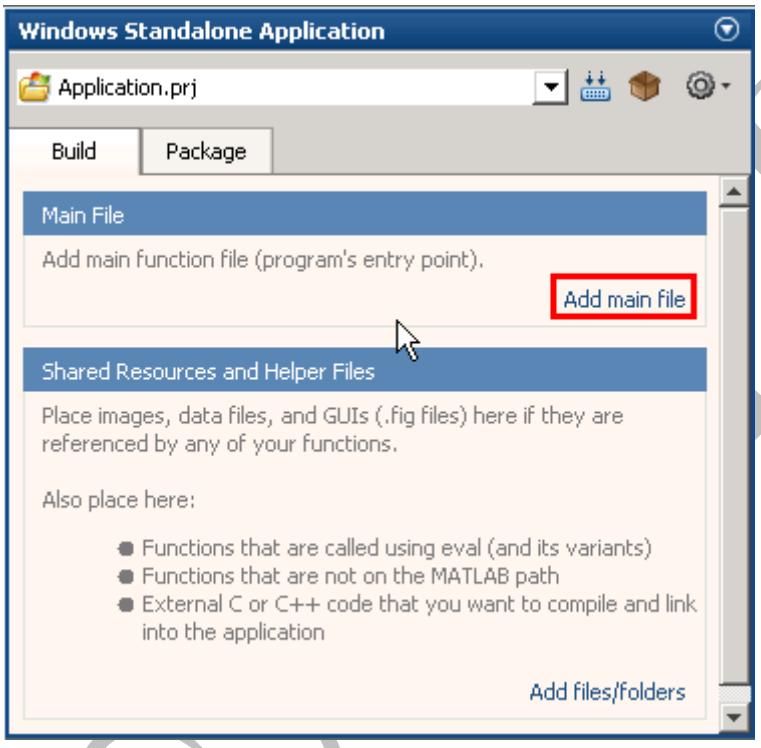
Create executable (.exe)*

You need MATLAB Runtime for this on the customer computer.

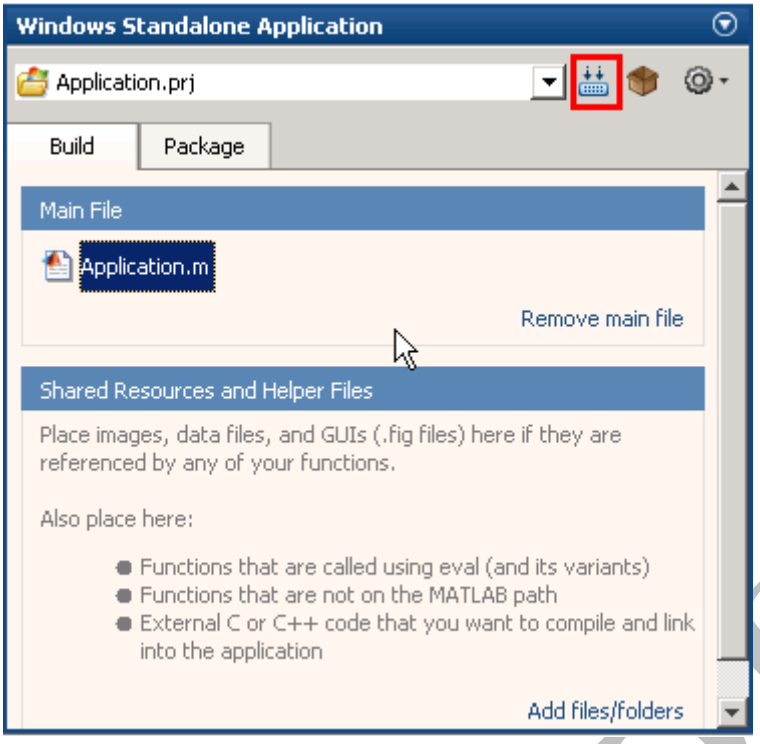


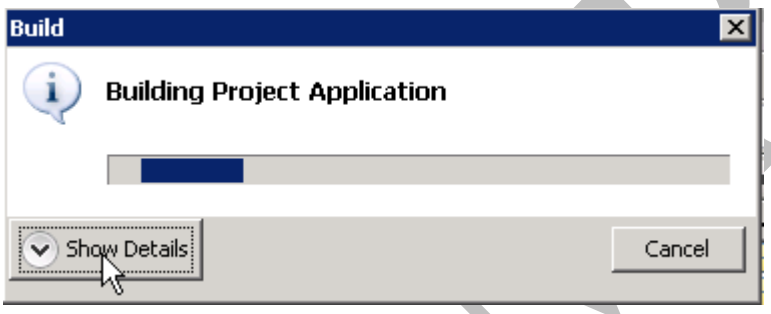


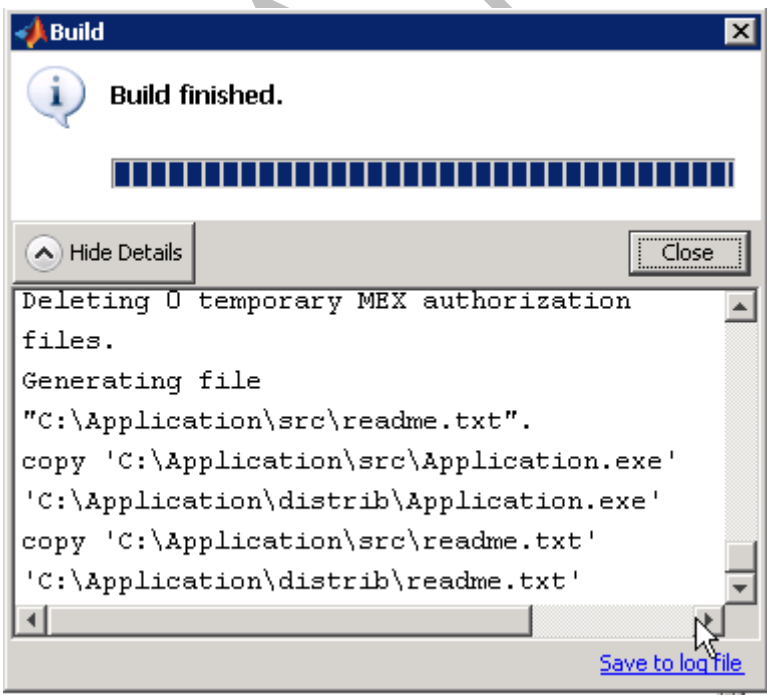

Deployment Project
New
Open
Name:
Application.prj
Location:
C:\
...
Type:
Windows Standalone Application
OK
Cancel

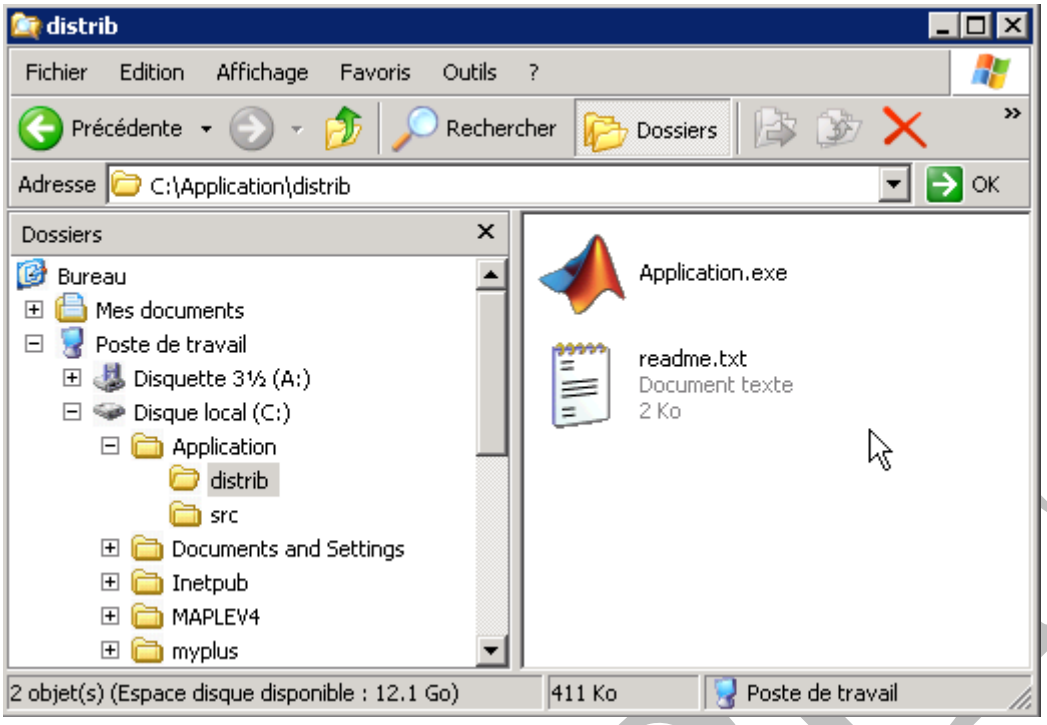

Windows Standalone Application
Application.prj
Build
Package
Main File
Add main function file (program's entry point).
Add main file
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.
Also place here:
Functions that are called using eval (and its variants)
Functions that are not on the MATLAB path
External C or C++ code that you want to compile and link into the application
Add files/folders

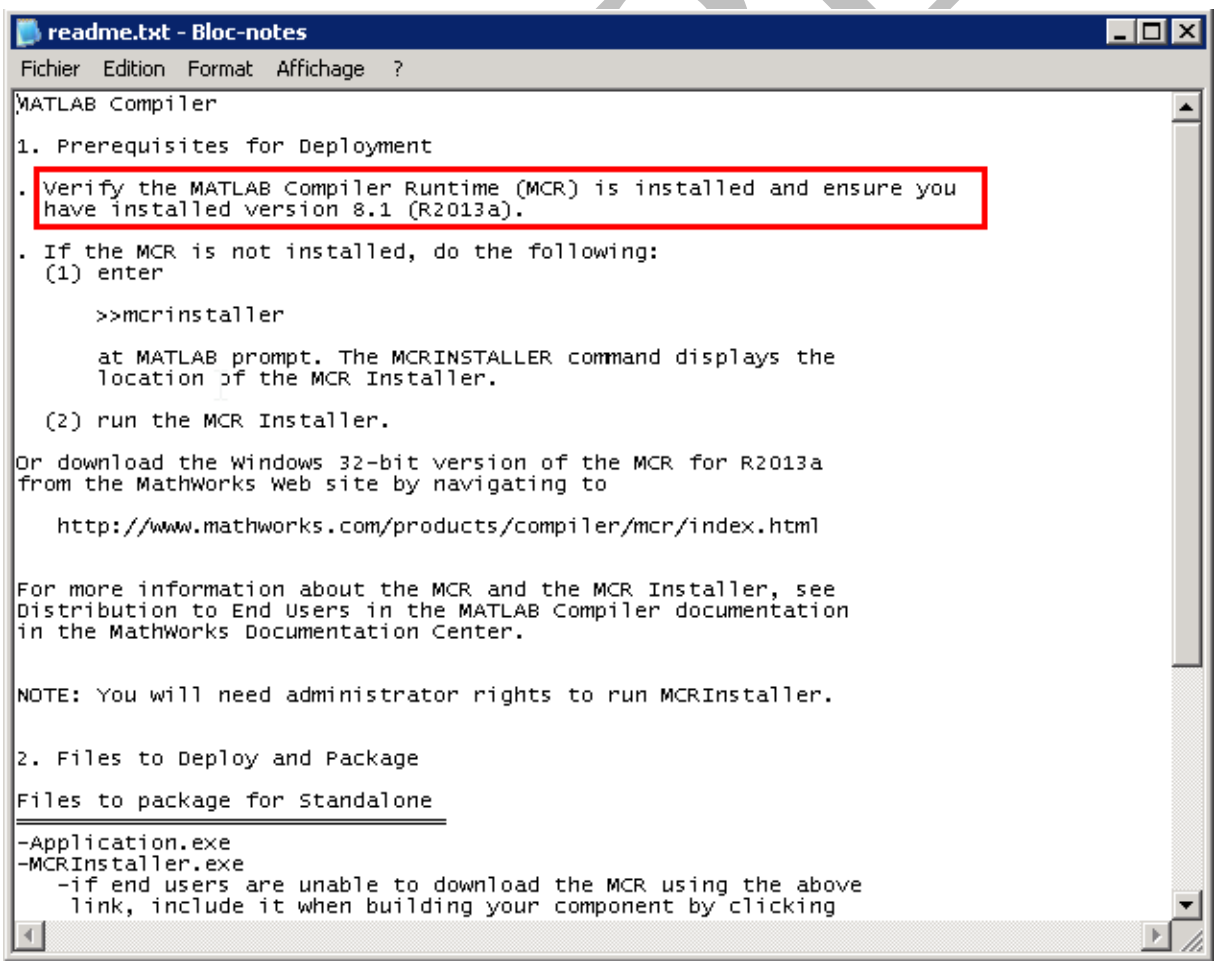
SAMPLE

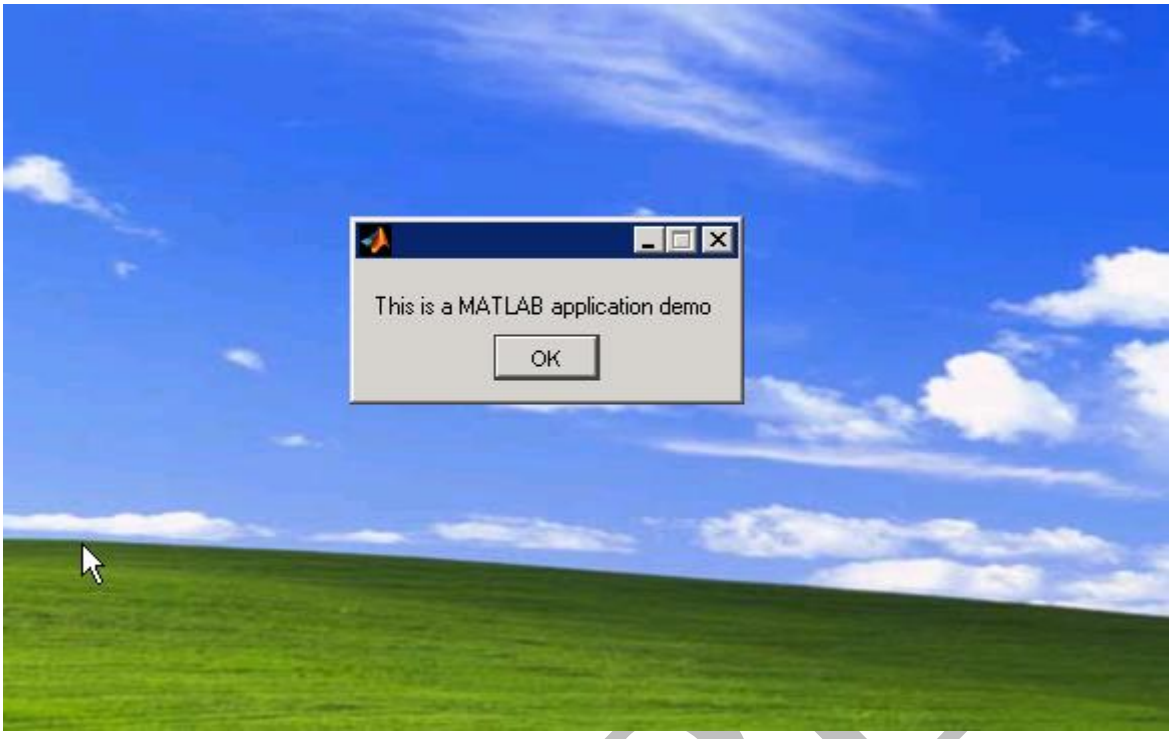

Windows Standalone Application
Application.prj
Build
Package
Main File
Application.m
Remove main file
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.
Also place here:
Functions that are called using eval (and its variants)
Functions that are not on the MATLAB path
External C or C++ code that you want to compile and link into the application
Add files/folders


Build
Building Project Application
Show Details
Cancel


Build
Build finished.
Hide Details
Close
Deleting 0 temporary MEX authorization files.
Generating file "C:\Application\src\readme.txt".
copy 'C:\Application\src\Application.exe' 'C:\Application\distrib\Application.exe'
copy 'C:\Application\src\readme.txt' 'C:\Application\distrib\readme.txt'
Save to log file

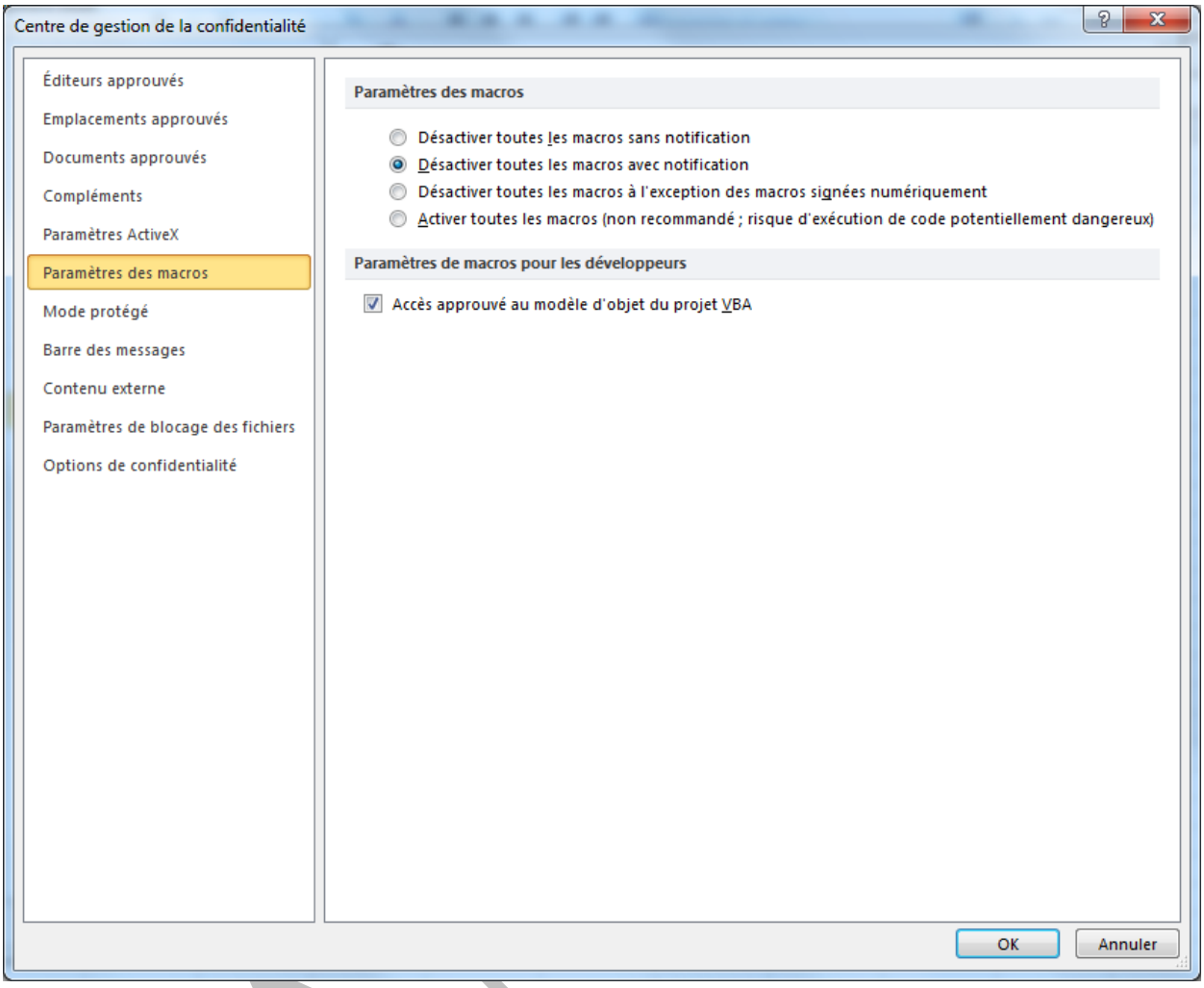

distrib
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Dossiers
Adresse C:\Application\distrib
OK
Dossiers
Bureau
Mes documents
Poste de travail
Disquette 3½ (A:)
Disque local (C:)
Application
distrib
src
Documents and Settings
Inetpub
MAPLEV4
myplus
Application.exe
readme.txt
Document texte
2 Ko
2 objet(s) (Espace disque disponible : 12.1 Go)
411 Ko
Poste de travail

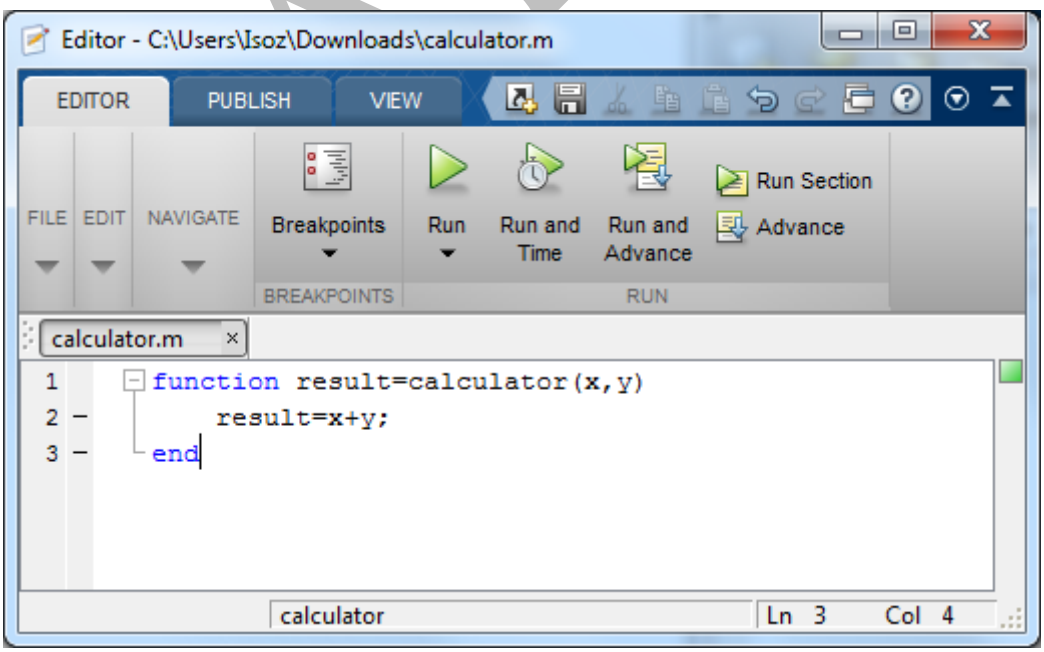

readme.txt - Bloc-notes
Fichier Edition Format Affichage ?
MATLAB Compiler
1. Prerequisites for Deployment
. Verify the MATLAB Compiler Runtime (MCR) is installed and ensure you have installed version 8.1 (R2013a).
. If the MCR is not installed, do the following:
(1) enter
>>mcrinstaller
at MATLAB prompt. The MCRINSTALLER command displays the location of the MCR Installer.
(2) run the MCR Installer.
Or download the Windows 32-bit version of the MCR for R2013a from the MathWorks Web site by navigating to
http://www.mathworks.com/products/compiler/mcr/index.html
For more information about the MCR and the MCR Installer, see Distribution to End Users in the MATLAB Compiler documentation in the MathWorks Documentation Center.
NOTE: You will need administrator rights to run MCRInstaller.
2. Files to Deploy and Package
Files to package for Standalone
-Application.exe
-MCRInstaller.exe
-if end users are unable to download the MCR using the above link, include it when building your component by clicking

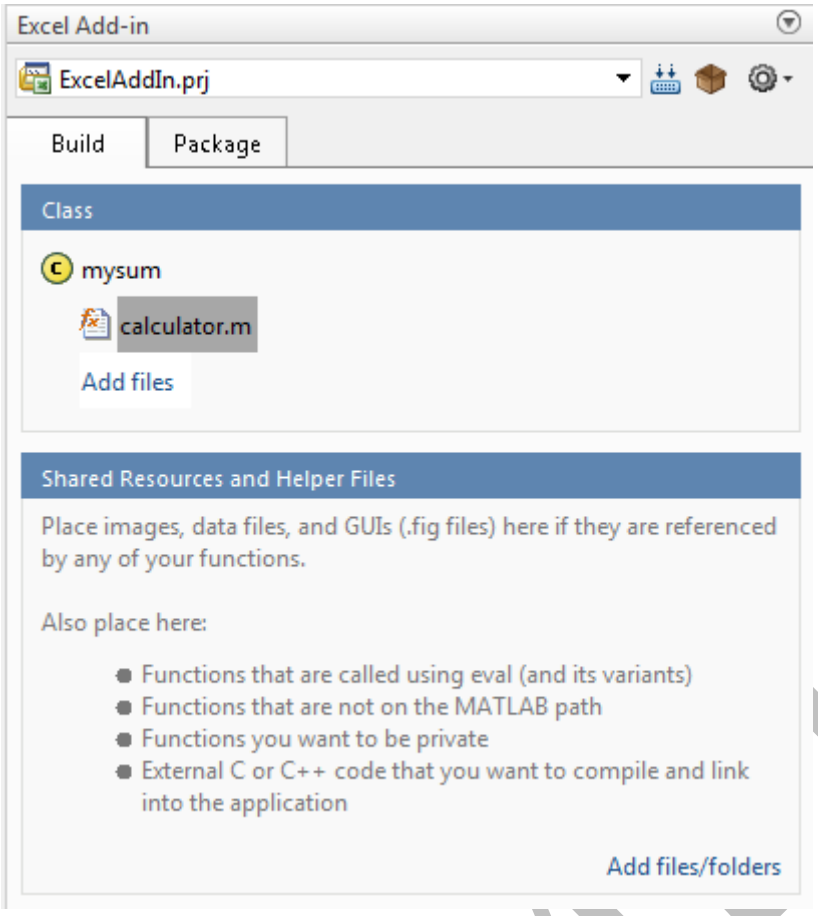


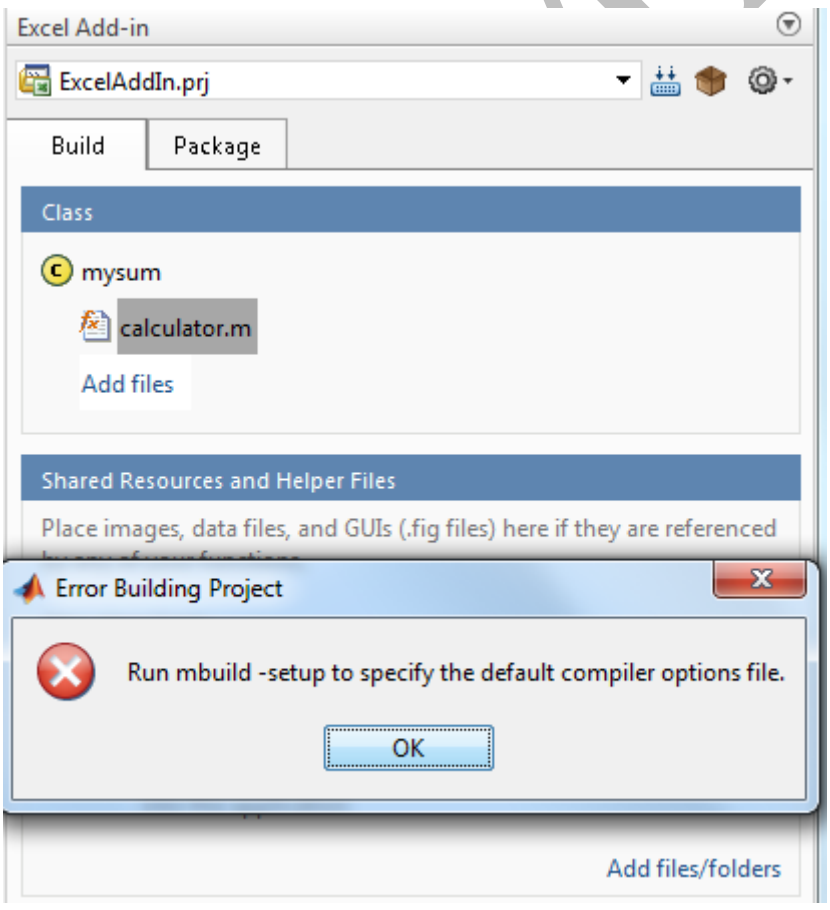
SAMPLE

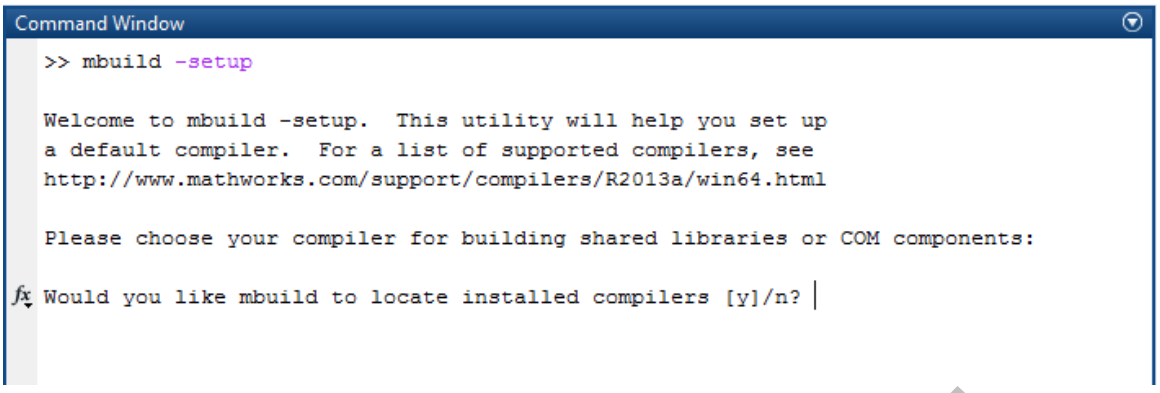
Call MATLAB from Microsoft Excel

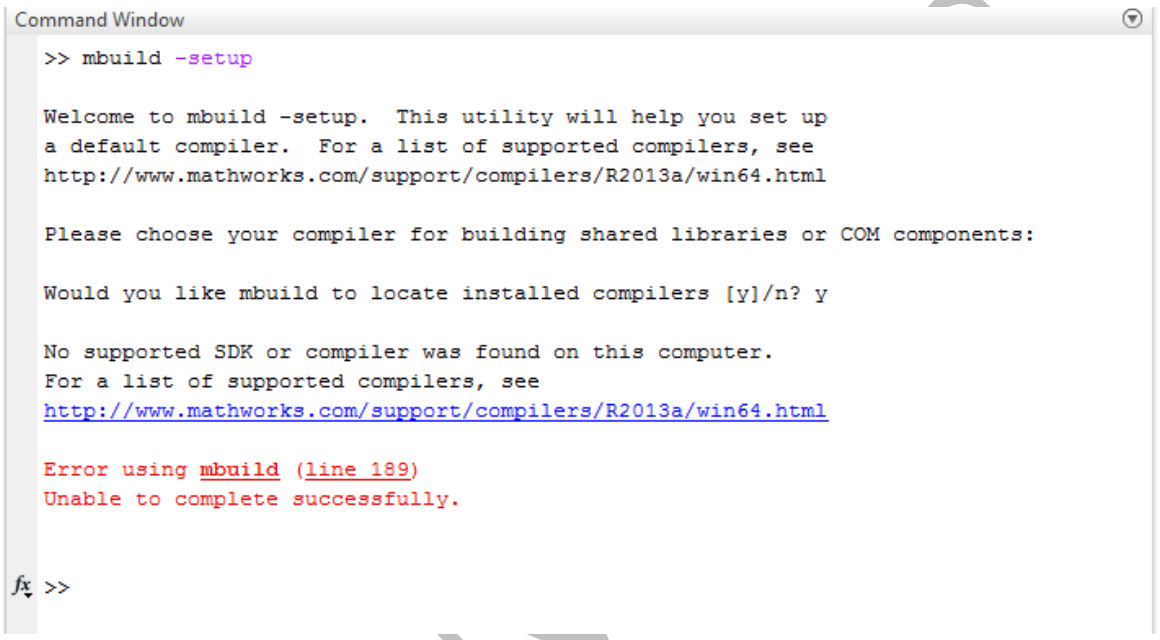




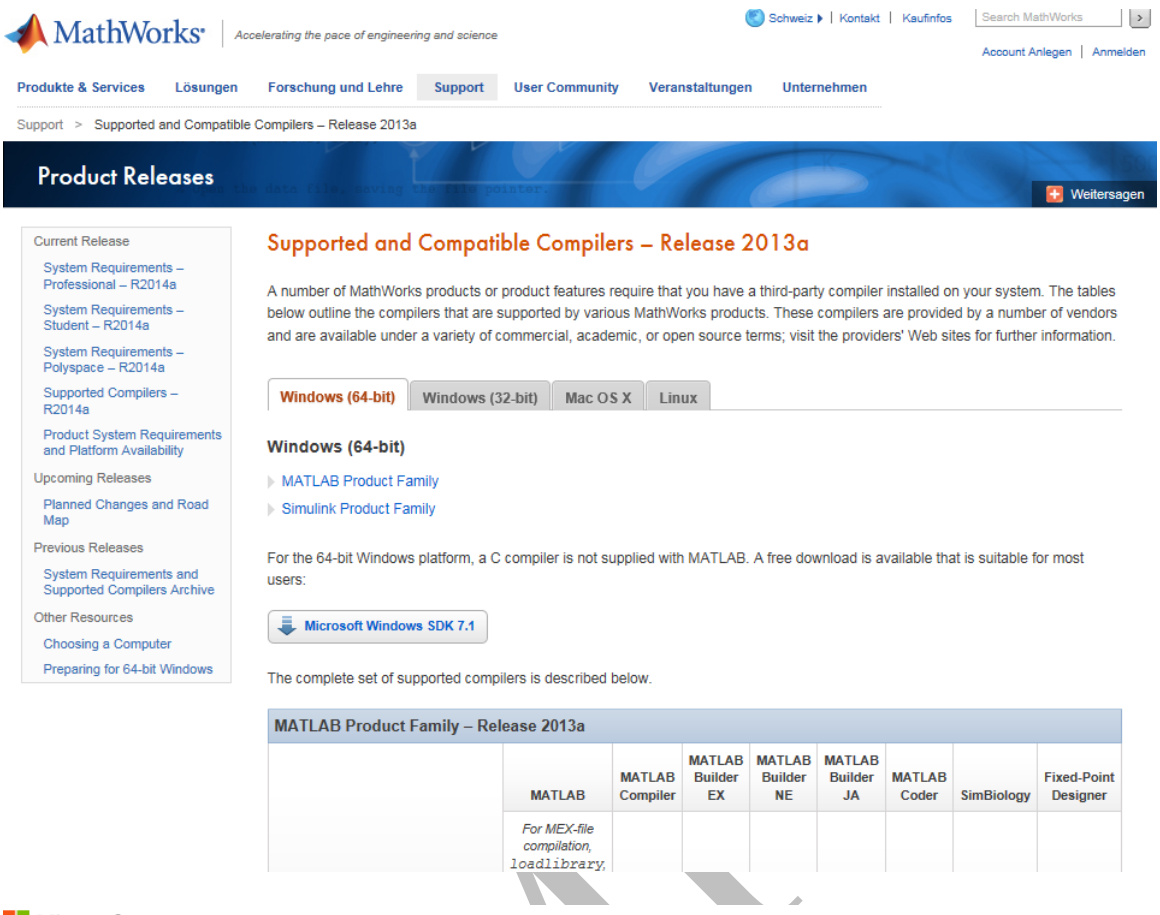

Excel Add-in
ExcelAddIn.prj
Build
Package
Class
mysum
calculator.m
Add files
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.
Also place here:
Functions that are called using eval (and its variants)
Functions that are not on the MATLAB path
Functions you want to be private
External C or C++ code that you want to compile and link into the application
Add files/folders

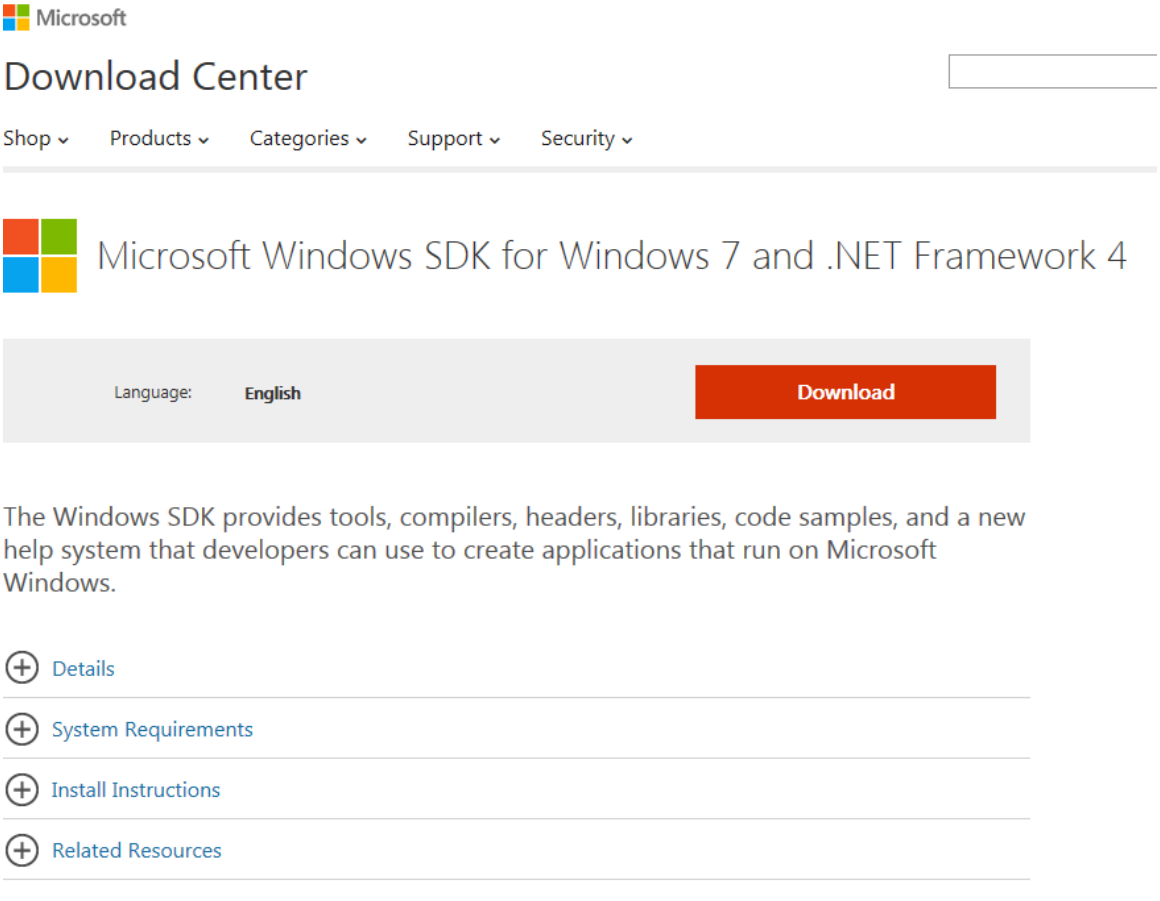

Excel Add-in
ExcelAddIn.prj
Build
Package
Class
mysum
calculator.m
Add files
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced
Error Building Project
Run mbuild -setup to specify the default compiler options file.
OK
Add files/folders

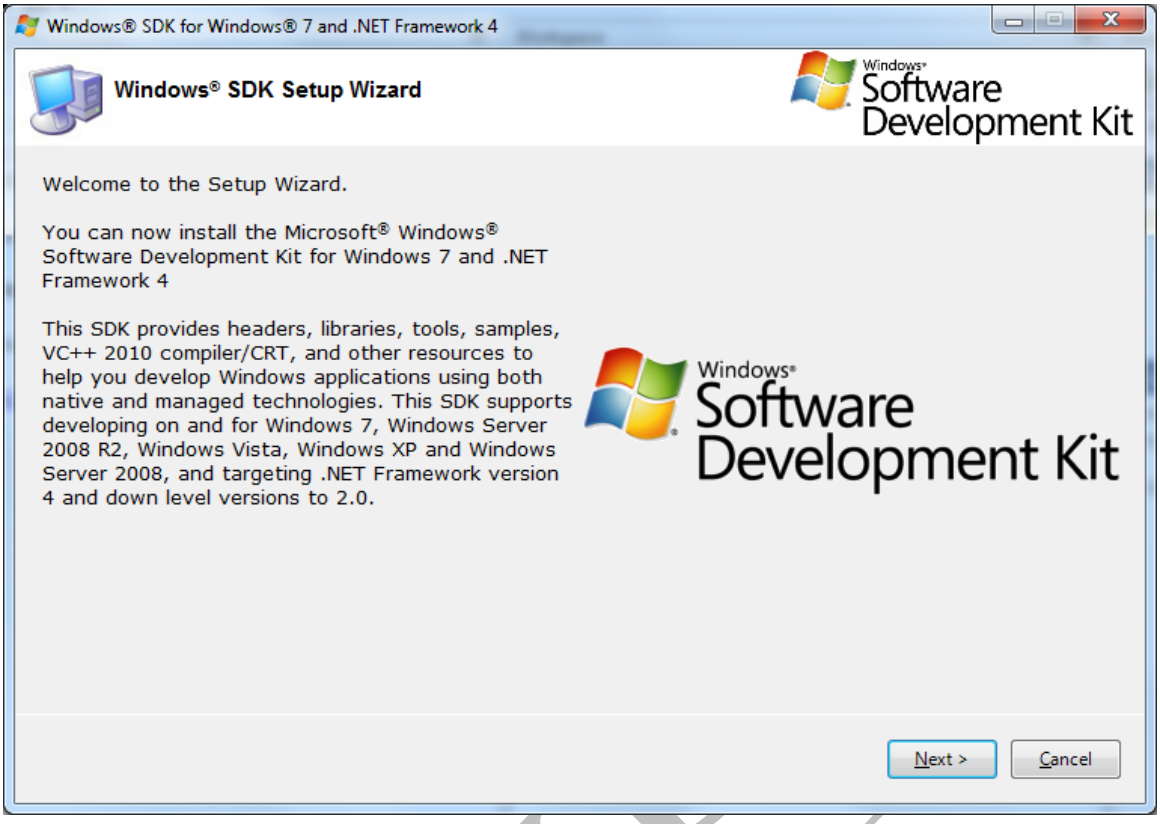

Command Window
>> mbuild -setup
Welcome to mbuild -setup. This utility will help you set up
a default compiler. For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2013a/win64.html
Please choose your compiler for building shared libraries or COM components:
Would you like mbuild to locate installed compilers [y]/n?


Command Window
>> mbuild -setup
Welcome to mbuild -setup. This utility will help you set up
a default compiler. For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2013a/win64.html
Please choose your compiler for building shared libraries or COM components:
Would you like mbuild to locate installed compilers [y]/n? y
No supported SDK or compiler was found on this computer.
For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2013a/win64.html
Error using mbuild (line 189)
Unable to complete successfully.
>>

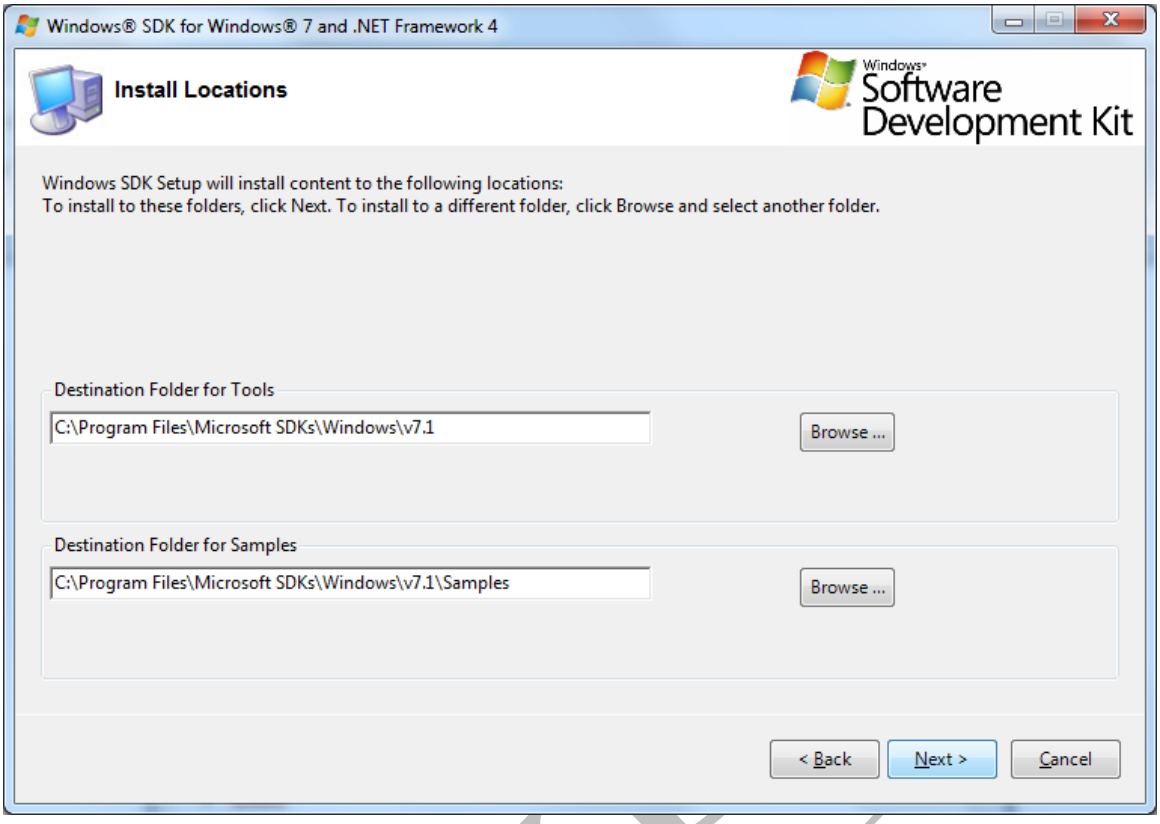
SAM

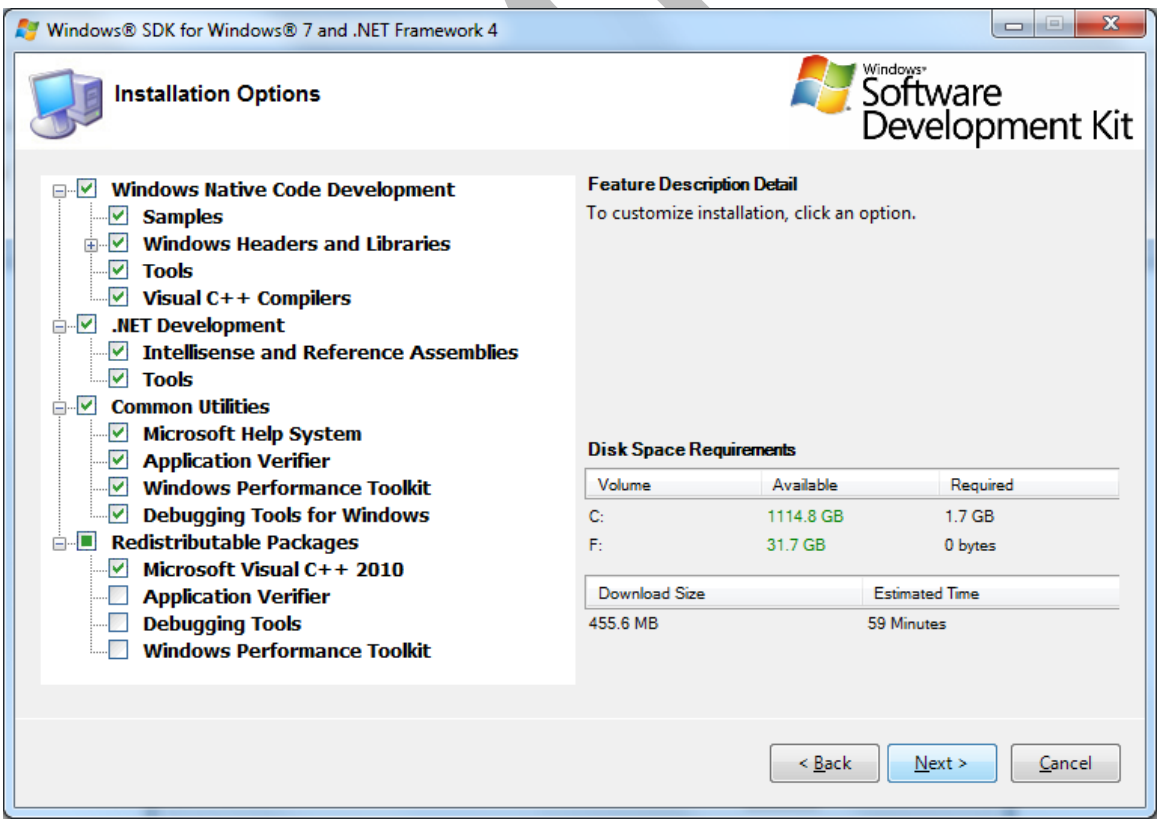

MathWorks
Accelerating the pace of engineering and science
Schweiz | Kontakt | Kaufinfos
Search MathWorks
Account Anlegen | Anmelden
Produkte & Services
Lösungen
Forschung und Lehre
Support
User Community
Veranstaltungen
Unternehmen
Support > Supported and Compatible Compilers – Release 2013a
Product Releases
Weitersagen
Current Release
System Requirements – Professional – R2014a
System Requirements – Student – R2014a
System Requirements – Polyspace – R2014a
Supported Compilers – R2014a
Product System Requirements and Platform Availability
Upcoming Releases
Planned Changes and Road Map
Previous Releases
System Requirements and Supported Compilers Archive
Other Resources
Choosing a Computer
Preparing for 64-bit Windows
Supported and Compatible Compilers – Release 2013a
A number of MathWorks products or product features require that you have a third-party compiler installed on your system. The tables below outline the compilers that are supported by various MathWorks products. These compilers are provided by a number of vendors and are available under a variety of commercial, academic, or open source terms; visit the providers' Web sites for further information.
Windows (64-bit)
Windows (32-bit)
Mac OS X
Linux
Windows (64-bit)
MATLAB Product Family
Simulink Product Family
For the 64-bit Windows platform, a C compiler is not supplied with MATLAB. A free download is available that is suitable for most users:
Microsoft Windows SDK 7.1
The complete set of supported compilers is described below.
MATLAB Product Family – Release 2013a
MATLAB
MATLAB Compiler
MATLAB Builder EX
MATLAB Builder NE
MATLAB Builder JA
MATLAB Coder
SimBiology
Fixed-Point Designer
For MEX-file compilation, loadlibrary,

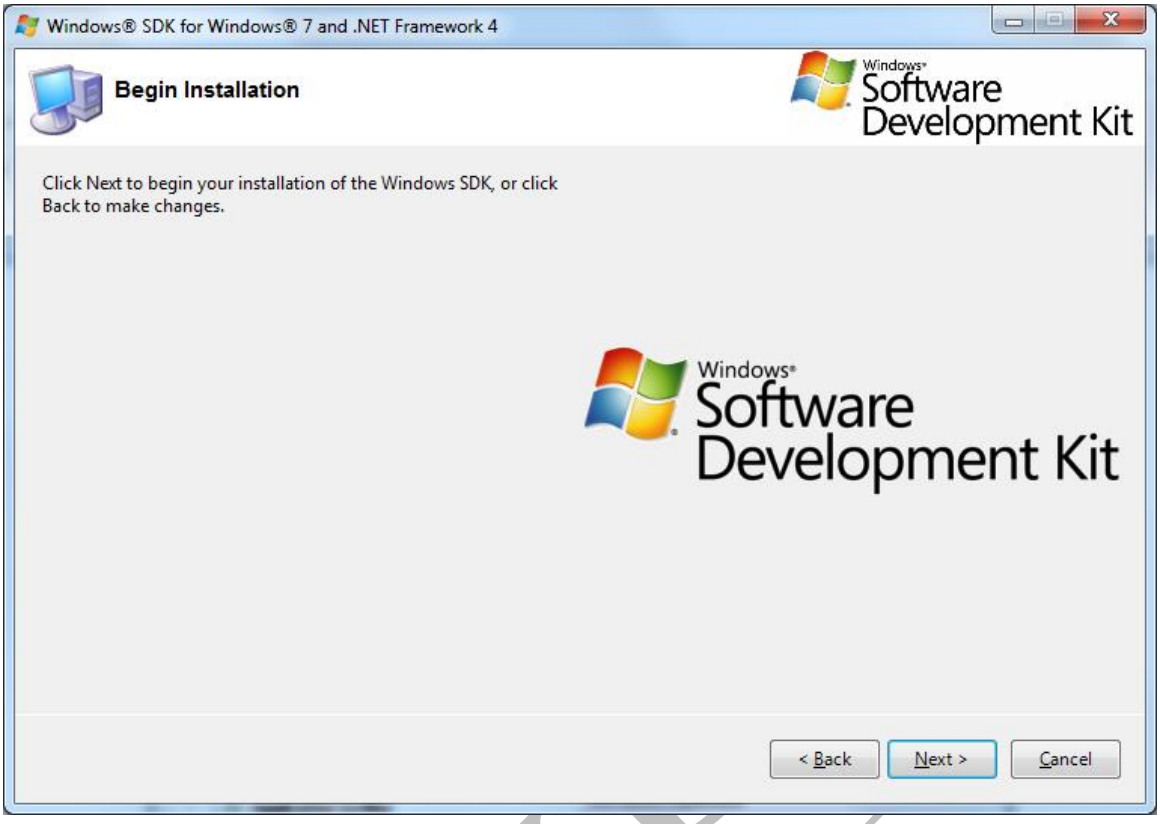

Microsoft
Download Center
Shop
Products
Categories
Support
Security
Microsoft Windows SDK for Windows 7 and .NET Framework 4
Language: English
Download
The Windows SDK provides tools, compilers, headers, libraries, code samples, and a new help system that developers can use to create applications that run on Microsoft Windows.
Details
System Requirements
Install Instructions
Related Resources

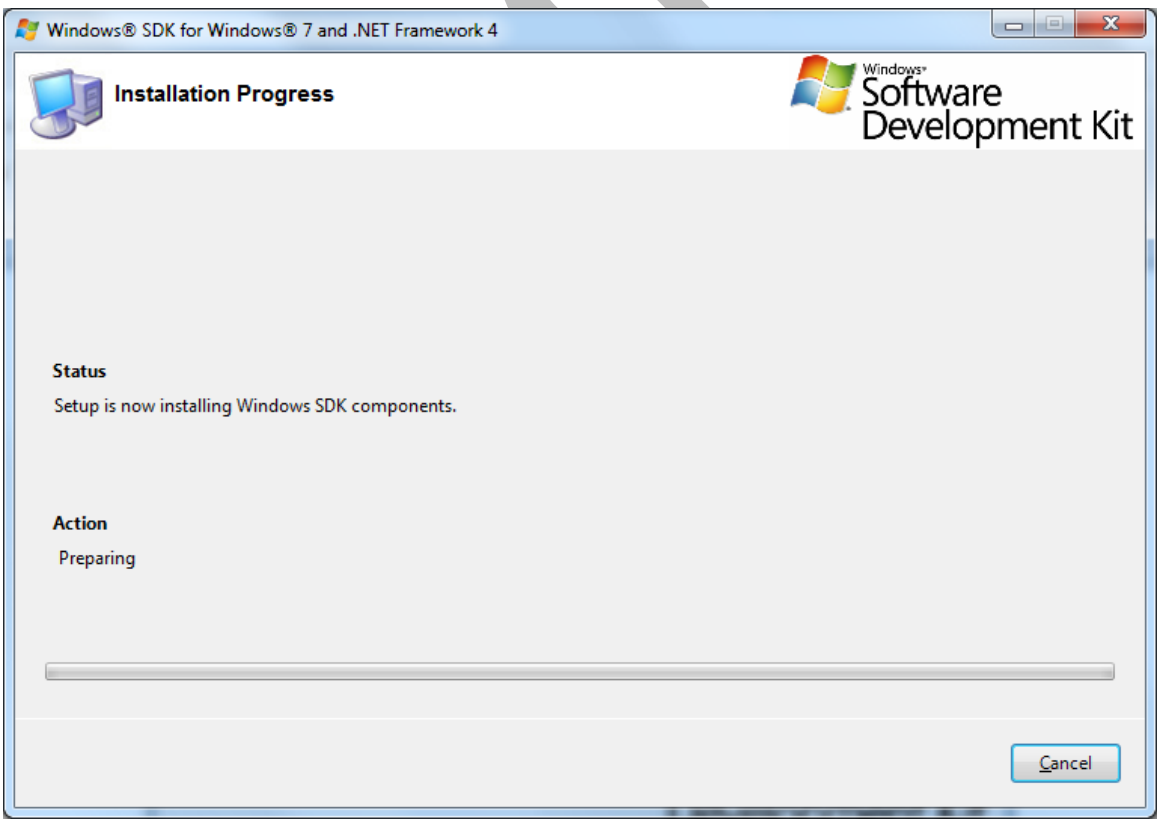

Windows® SDK for Windows® 7 and .NET Framework 4
Windows® SDK Setup Wizard
Windows Software Development Kit
Welcome to the Setup Wizard.
You can now install the Microsoft® Windows® Software Development Kit for Windows 7 and .NET Framework 4
This SDK provides headers, libraries, tools, samples, VC++ 2010 compiler/CRT, and other resources to help you develop Windows applications using both native and managed technologies. This SDK supports developing on and for Windows 7, Windows Server 2008 R2, Windows Vista, Windows XP and Windows Server 2008, and targeting .NET Framework version 4 and down level versions to 2.0.
Windows Software Development Kit
Next >
Cancel

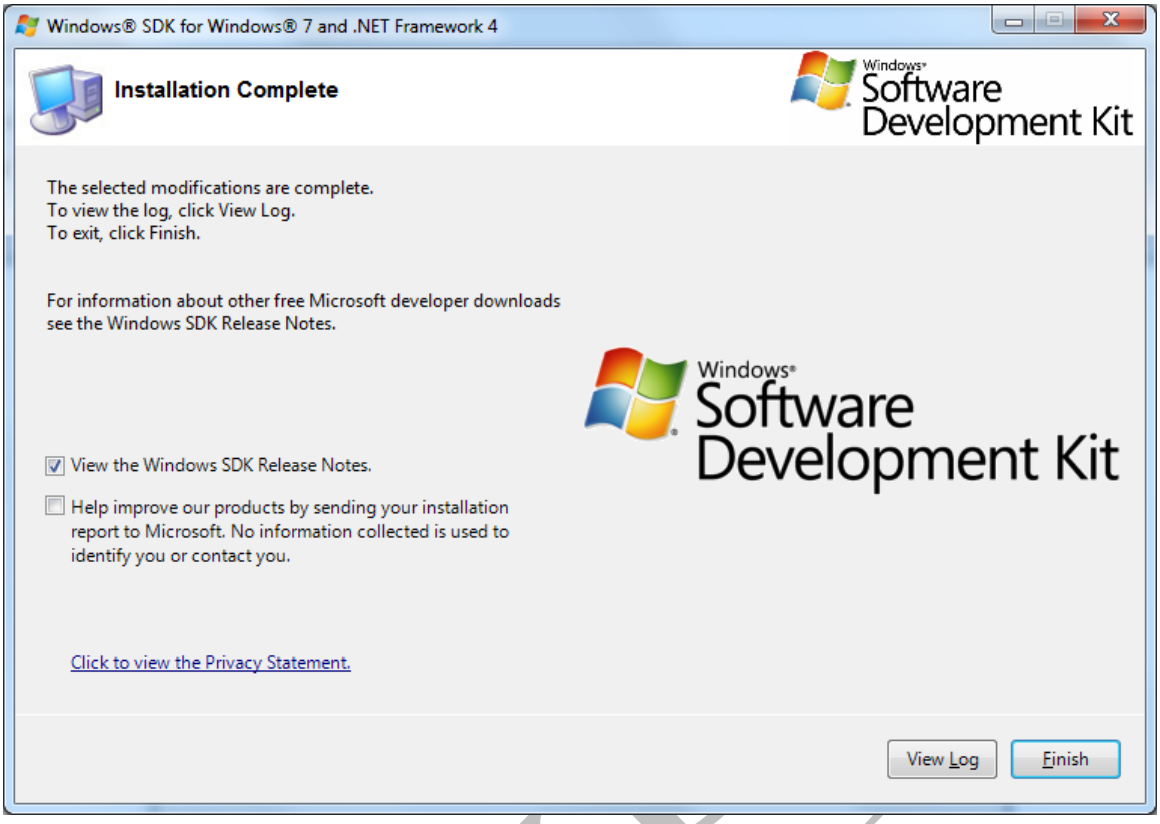

Windows® SDK for Windows® 7 and .NET Framework 4
End-User License Agreement
Windows Software Development Kit
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT WINDOWS SOFTWARE DEVELOPMENT KIT FOR WINDOWS 7 and .NET FRAMEWORK 4
These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft
updates,
supplements,
Please read the End-User License Agreement carefully and select either "I Agree" or "I Disagree."
I Agree
I Disagree
Print
< Back
Next >
Cancel


Windows® SDK for Windows® 7 and .NET Framework 4
Install Locations
Windows Software Development Kit
Windows SDK Setup will install content to the following locations:
To install to these folders, click Next. To install to a different folder, click Browse and select another folder.
Destination Folder for Tools
C:\Program Files\Microsoft SDKs\Windows\v7.1
Browse ...
Destination Folder for Samples
C:\Program Files\Microsoft SDKs\Windows\v7.1\Samples
Browse ...
< Back
Next >
Cancel

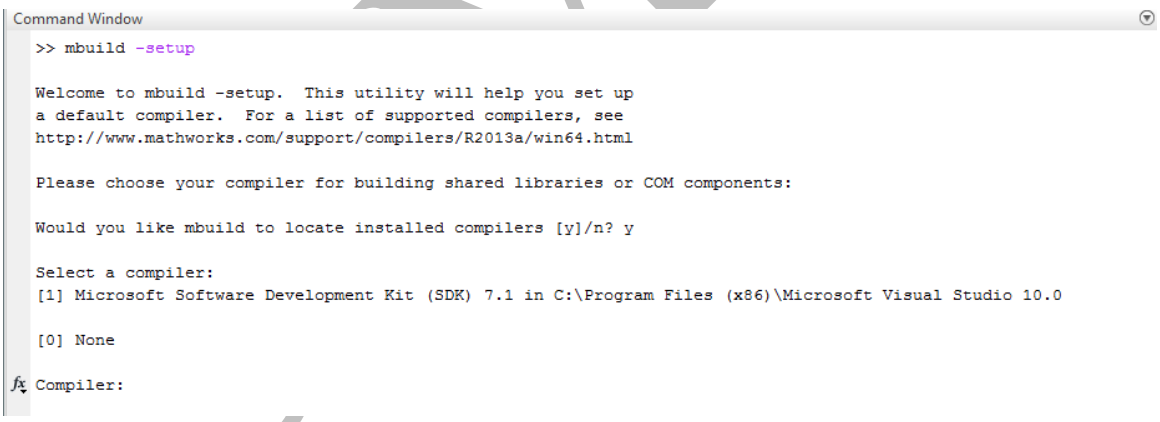

Windows® SDK for Windows® 7 and .NET Framework 4
Installation Options
Windows Software Development Kit
Windows Native Code Development
Samples
Windows Headers and Libraries
Tools
Visual C++ Compilers
.NET Development
Intellisense and Reference Assemblies
Tools
Common Utilities
Microsoft Help System
Application Verifier
Windows Performance Toolkit
Debugging Tools for Windows
Redistributable Packages
Microsoft Visual C++ 2010
Application Verifier
Debugging Tools
Windows Performance Toolkit
Feature Description Detail
To customize installation, click an option.
Disk Space Requirements
Volume
Available
Required
C:
1114.8 GB
1.7 GB
F:
31.7 GB
0 bytes
Download Size
Estimated Time
455.6 MB
59 Minutes
< Back
Next >
Cancel

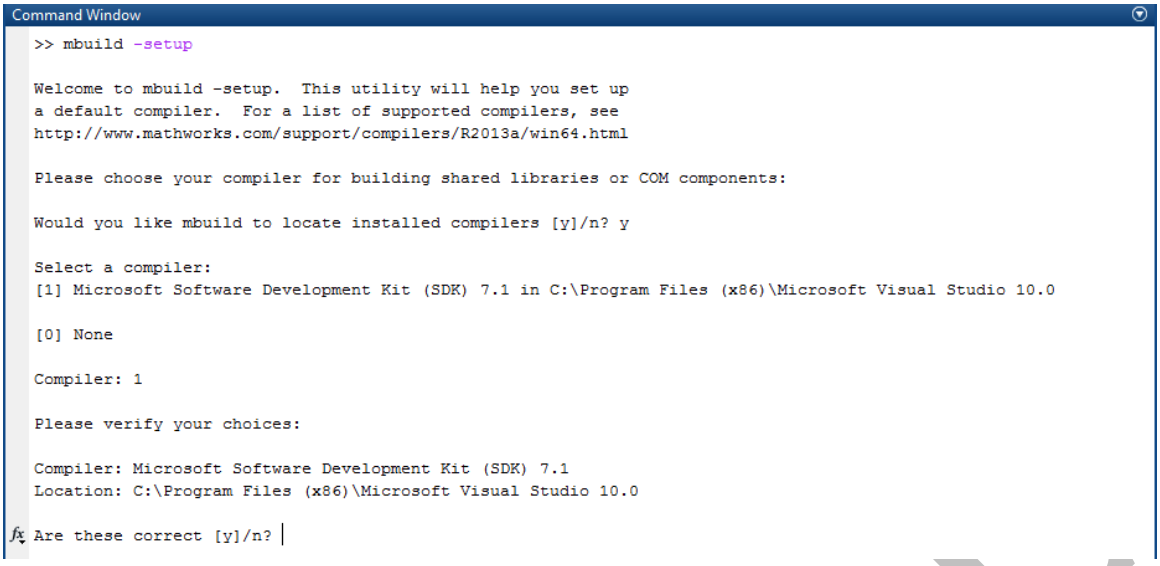

Windows® SDK for Windows® 7 and .NET Framework 4
Begin Installation
Windows Software Development Kit
Click Next to begin your installation of the Windows SDK, or click Back to make changes.
Windows Software Development Kit
< Back
Next >
Cancel

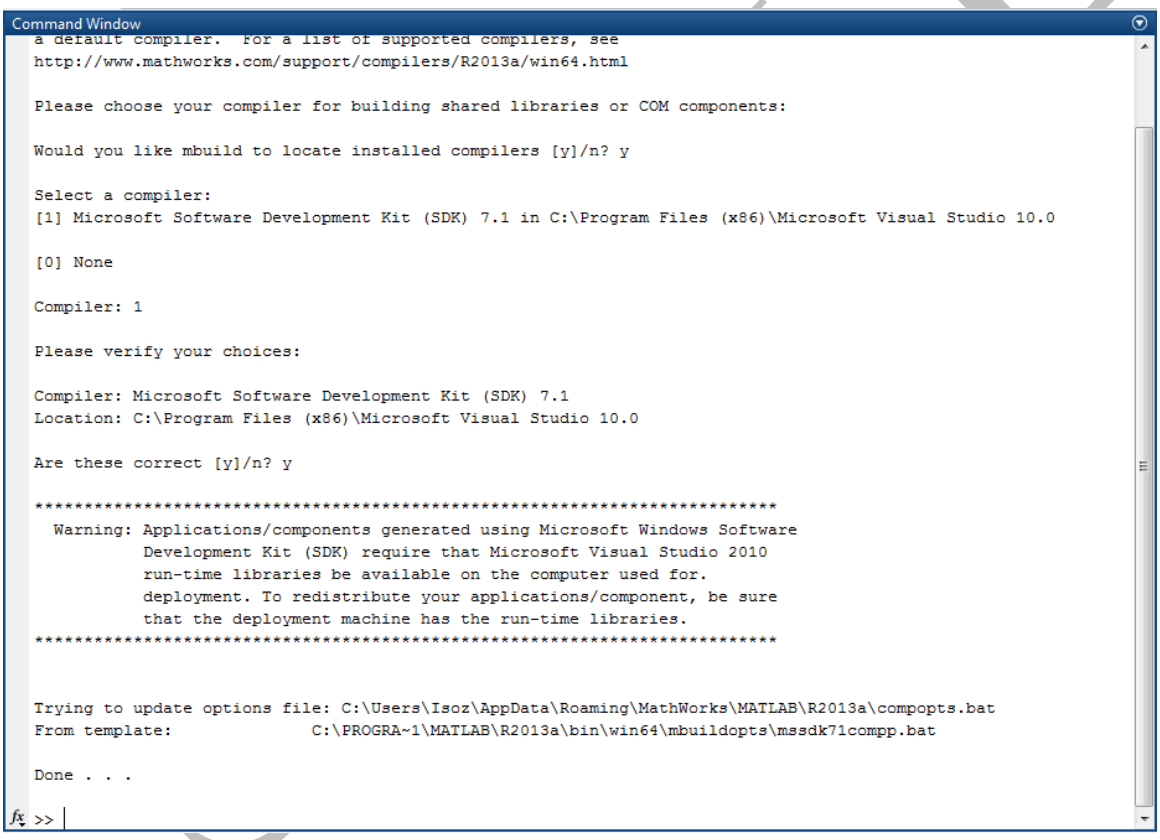

Windows® SDK for Windows® 7 and .NET Framework 4
Installation Progress
Windows Software Development Kit
Status
Setup is now installing Windows SDK components.
Action
Preparing
Cancel



You need to have the .Net Framework 4.0 the be able to run the following steps with MATLAB 2013 !! Remove further versions before, install .Net Framework 4.0 and reboot computer!

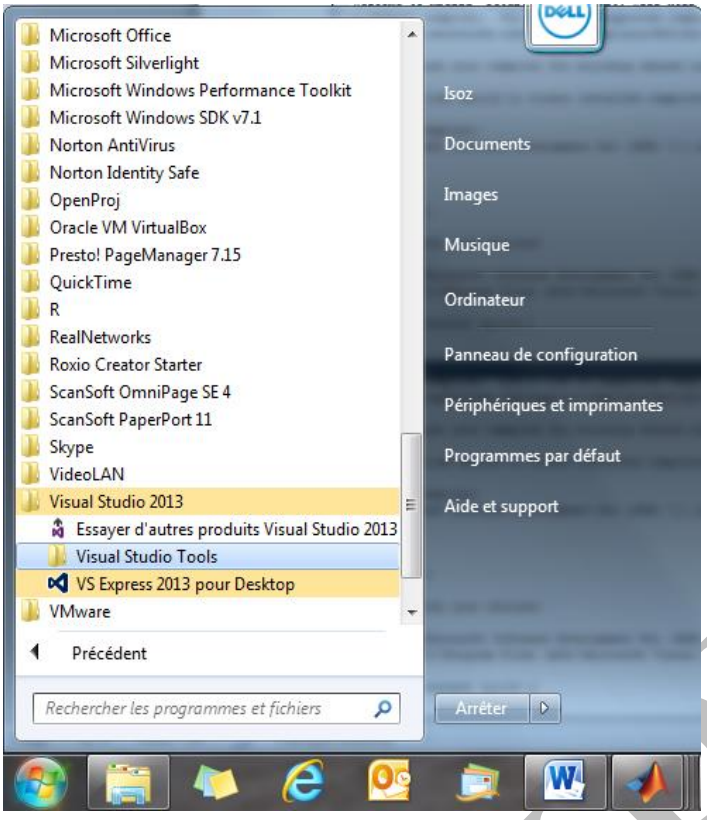


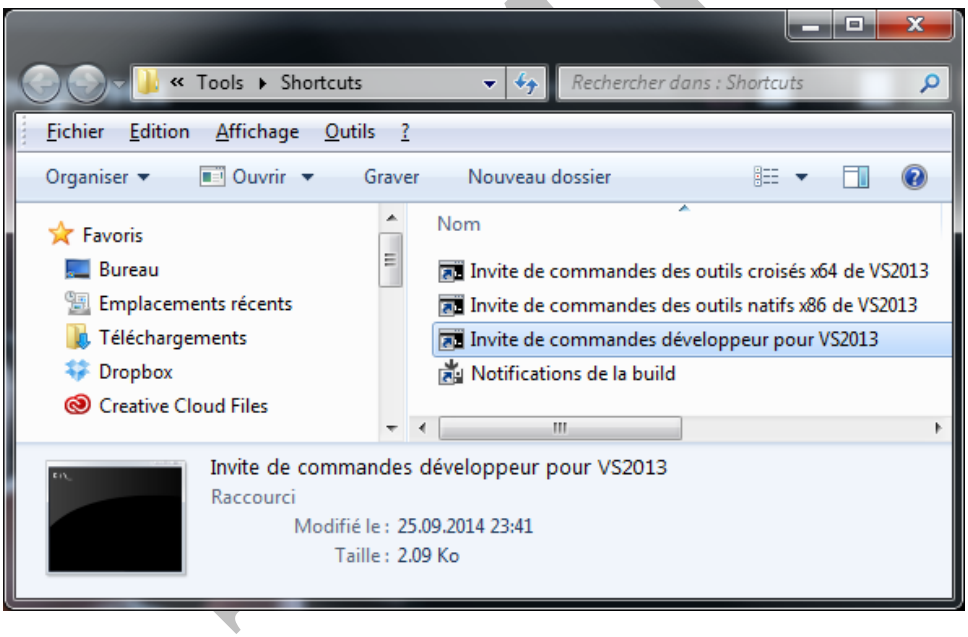


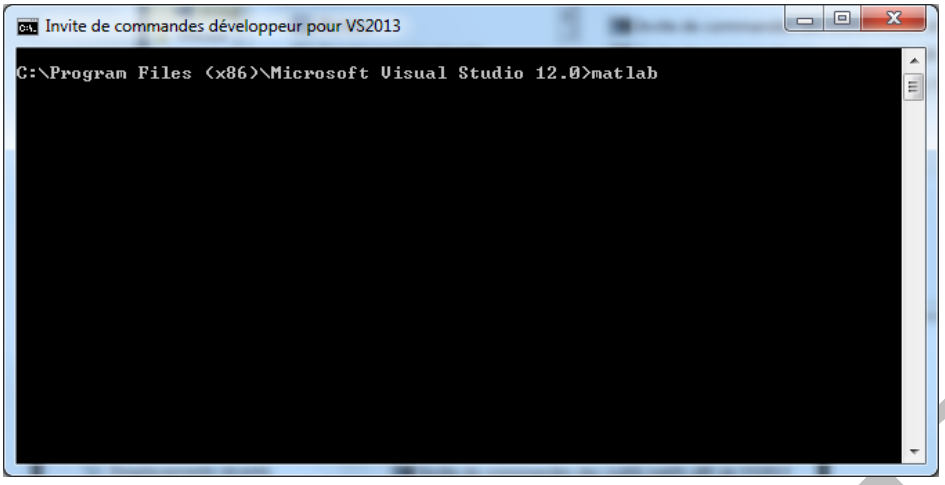


Install Also the last version of Visual Studio (Express or Not with all updates).

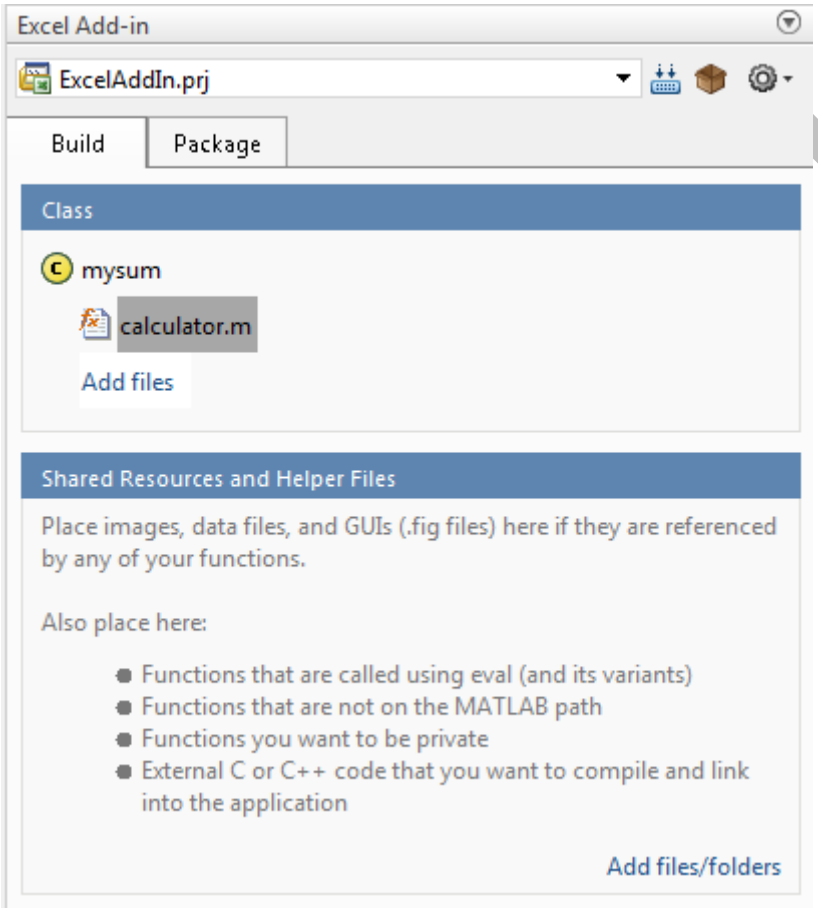
You must not launch MATLAB as normally if you want to be sure to not encounter any problem during the compilation.

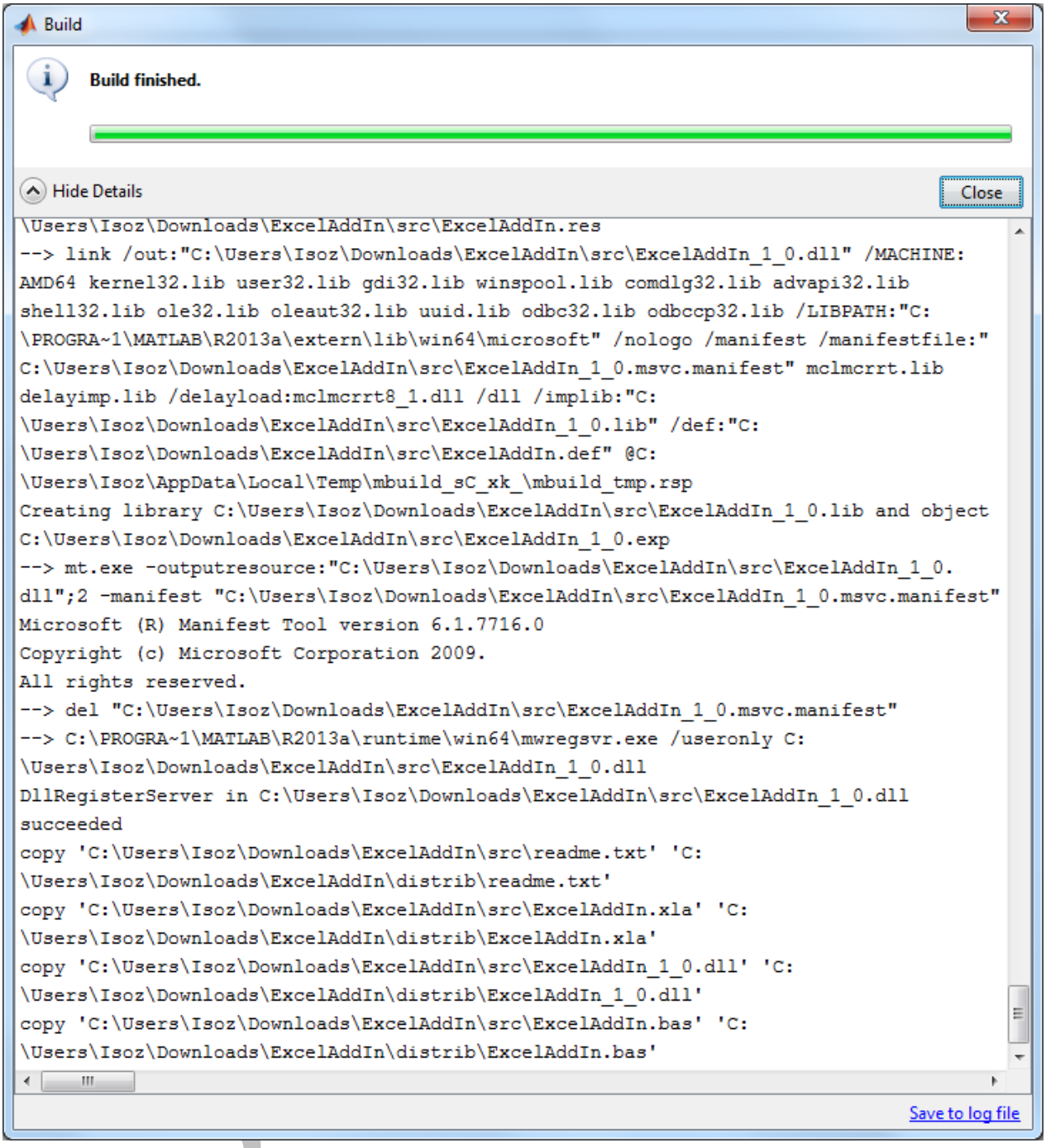

Microsoft Office
Microsoft Silverlight
Microsoft Windows Performance Toolkit
Microsoft Windows SDK v7.1
Norton AntiVirus
Norton Identity Safe
OpenProj
Oracle VM VirtualBox
Presto! PageManager 7.15
QuickTime
R
RealNetworks
Roxio Creator Starter
ScanSoft OmniPage SE 4
ScanSoft PaperPort 11
Skype
VideoLAN
Visual Studio 2013
Essayer d'autres produits Visual Studio 2013
Visual Studio Tools
VS Express 2013 pour Desktop
VMware
Précédent
Rechercher les programmes et fichiers
Isoz
Documents
Images
Musique
Ordinateur
Panneau de configuration
Périphériques et imprimantes
Programmes par défaut
Aide et support
Arrêter

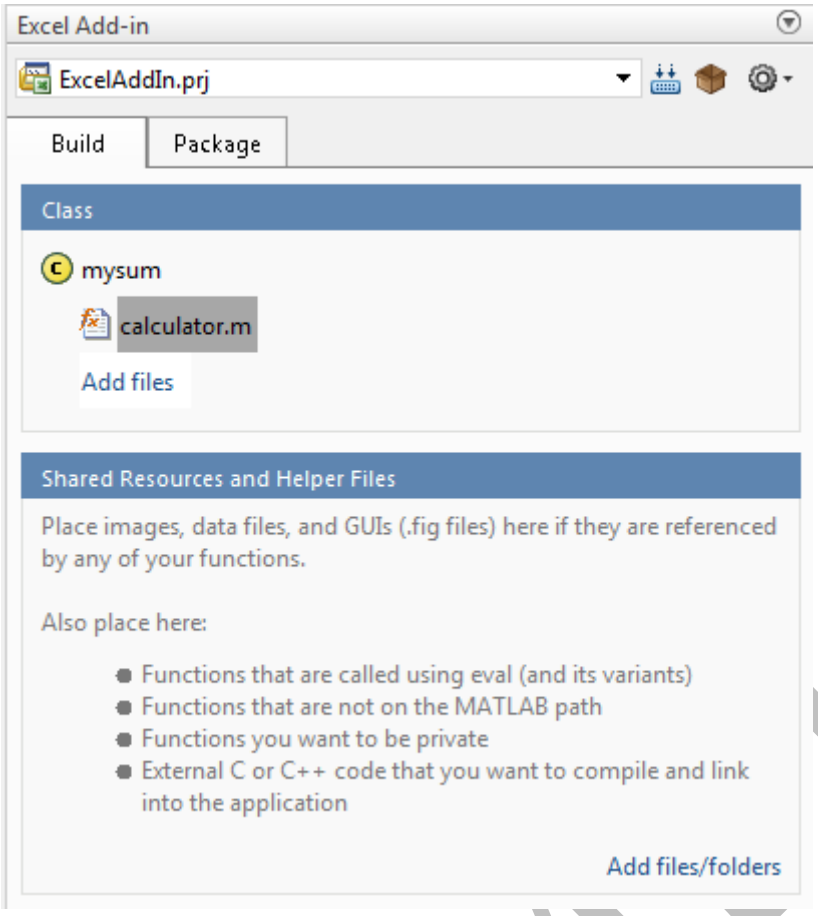

« Tools ▸ Shortcuts
Rechercher dans : Shortcuts
Fichier Edition Affichage Outils ?
Organiser
Ouvrir
Graver
Nouveau dossier
Favoris
Bureau
Emplacements récents
Téléchargements
Dropbox
Creative Cloud Files
Nom
Invite de commandes des outils croisés x64 de VS2013
Invite de commandes des outils natifs x86 de VS2013
Invite de commandes développeur pour VS2013
Notifications de la build
Invite de commandes développeur pour VS2013
Raccourci
Modifié le : 25.09.2014 23:41
Taille : 2.09 Ko



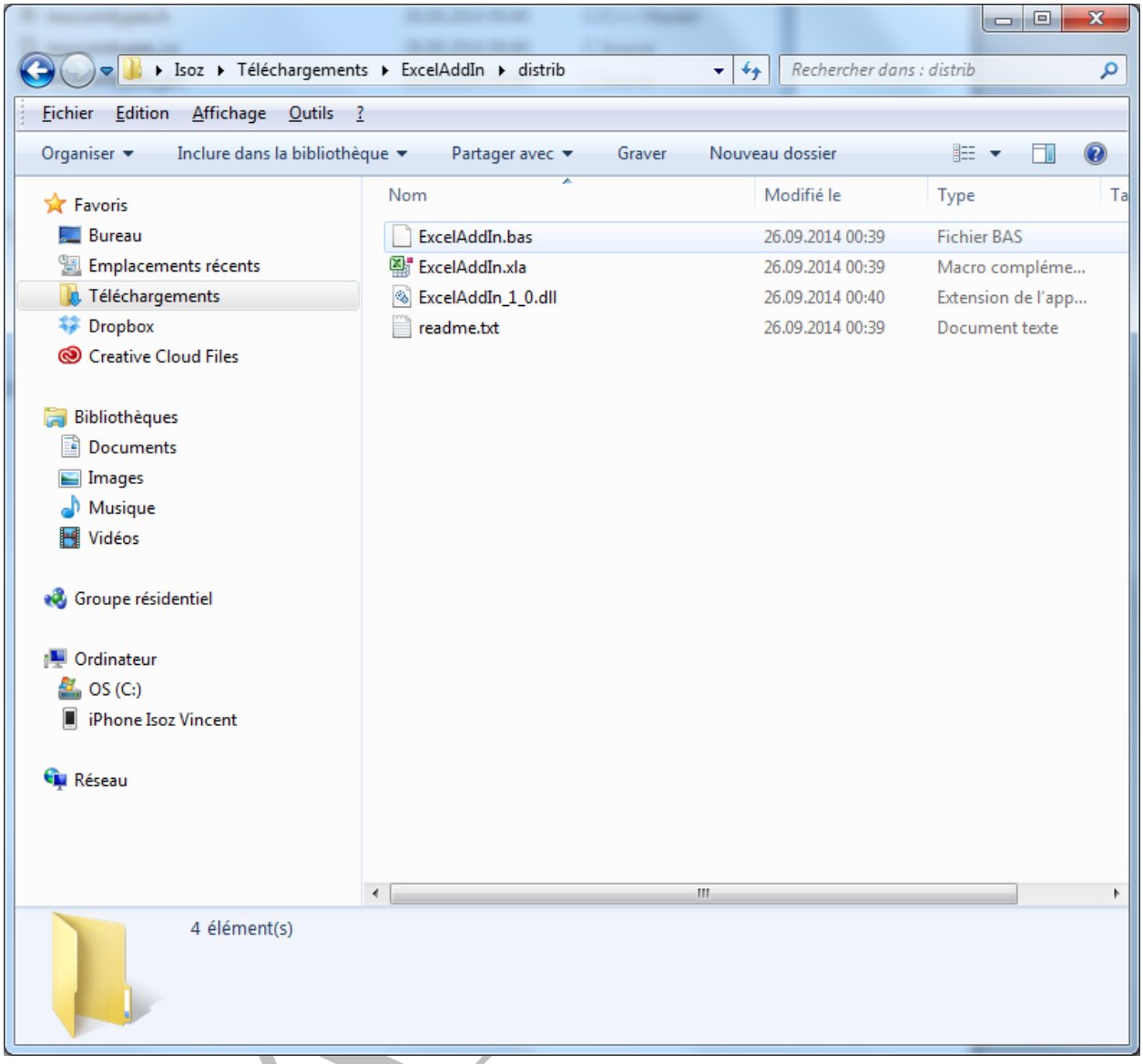
Once MATLAB open you restart the compiler and everything will work :



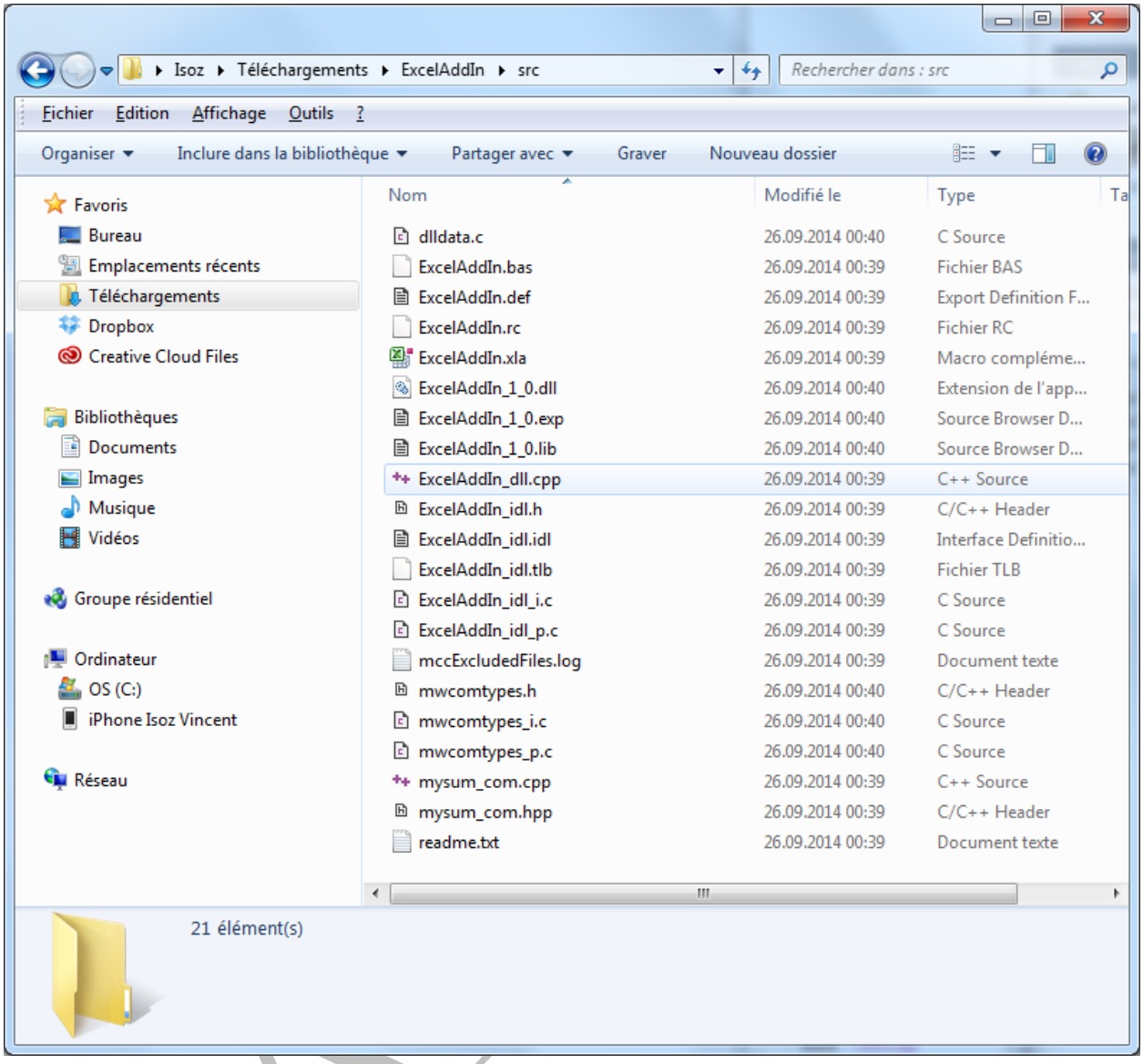

Build
Build finished.
Hide Details
Close
\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn.res
--> link /out:"C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.dll" /MACHINE:
AMD64 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /LIBPATH:"C:
\PROGRA~1\MATLAB\R2013a\extern\lib\win64\microsoft" /nologo /manifest /manifestfile:"
C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.msvc.manifest" mclmcrrt.lib
delayimp.lib /delayload:mclmcrrt8_1.dll /dll /implib:"C:
\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.lib" /def:"C:
\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn.def" @C:
\Users\Isoz\AppData\Local\Temp\mbuild_sC_xk_\mbuild_tmp.rsp
Creating library C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.lib and object
C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.exp
--> mt.exe -outputresource:"C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.
dll";2 -manifest "C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.msvc.manifest"
Microsoft (R) Manifest Tool version 6.1.7716.0
Copyright (c) Microsoft Corporation 2009.
All rights reserved.
--> del "C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.msvc.manifest"
--> C:\PROGRA~1\MATLAB\R2013a\runtime\win64\mwregsvr.exe /useronly C:
\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.dll
DllRegisterServer in C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.dll
succeeded
copy 'C:\Users\Isoz\Downloads\ExcelAddIn\src\readme.txt' 'C:
\Users\Isoz\Downloads\ExcelAddIn\distrib\readme.txt'
copy 'C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn.xla' 'C:
\Users\Isoz\Downloads\ExcelAddIn\distrib\ExcelAddIn.xla'
copy 'C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn_1_0.dll' 'C:
\Users\Isoz\Downloads\ExcelAddIn\distrib\ExcelAddIn_1_0.dll'
copy 'C:\Users\Isoz\Downloads\ExcelAddIn\src\ExcelAddIn.bas' 'C:
\Users\Isoz\Downloads\ExcelAddIn\distrib\ExcelAddIn.bas'
Save to log file


Excel Add-in
ExcelAddIn.prj
Build
Package
Class
mysum
calculator.m
Add files
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.
Also place here:
Functions that are called using eval (and its variants)
Functions that are not on the MATLAB path
Functions you want to be private
External C or C++ code that you want to compile and link into the application
Add files/folders

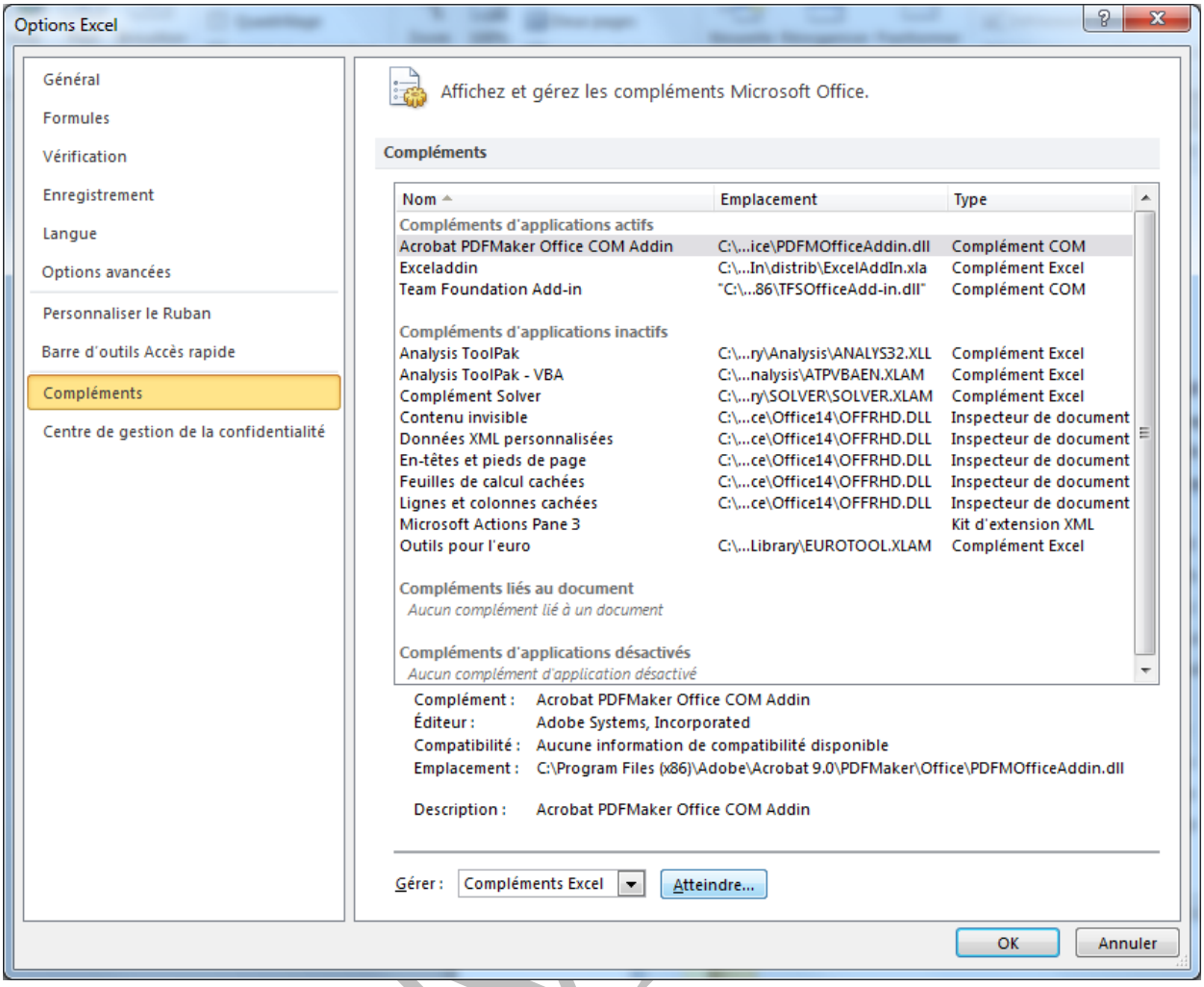
SAMPLE

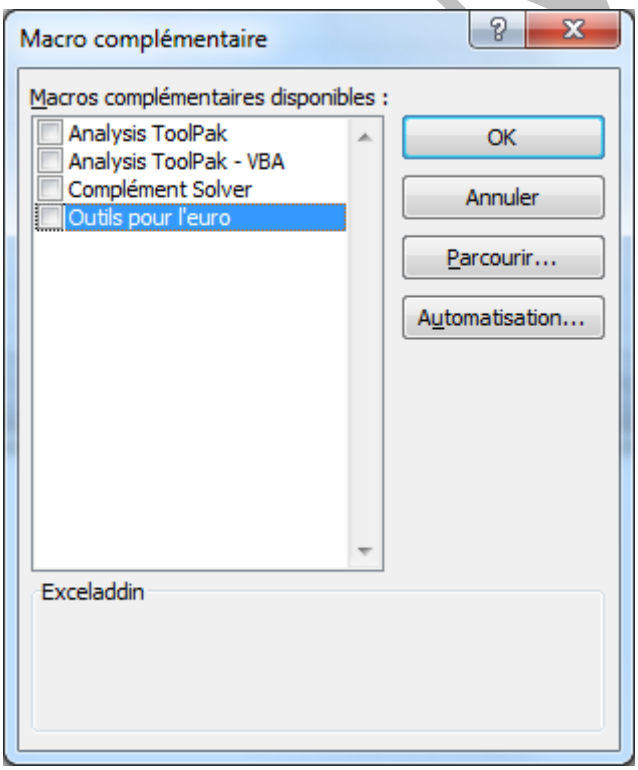

Isoz ▸ Téléchargements ▸ ExcelAddIn ▸ distrib
Rechercher dans : distrib
Fichier Edition Affichage Outils ?
Organiser
Inclure dans la bibliothèque
Partager avec
Graver
Nouveau dossier
Favoris
Bureau
Emplacements récents
Téléchargements
Dropbox
Creative Cloud Files
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
OS (C:)
iPhone Isoz Vincent
Réseau
Nom
Modifié le
Type
ExcelAddIn.bas 26.09.2014 00:39 Fichier BAS
ExcelAddIn.xla 26.09.2014 00:39 Macro compléme...
ExcelAddIn_1_0.dll 26.09.2014 00:40 Extension de l'app...
readme.txt 26.09.2014 00:39 Document texte
4 élément(s)

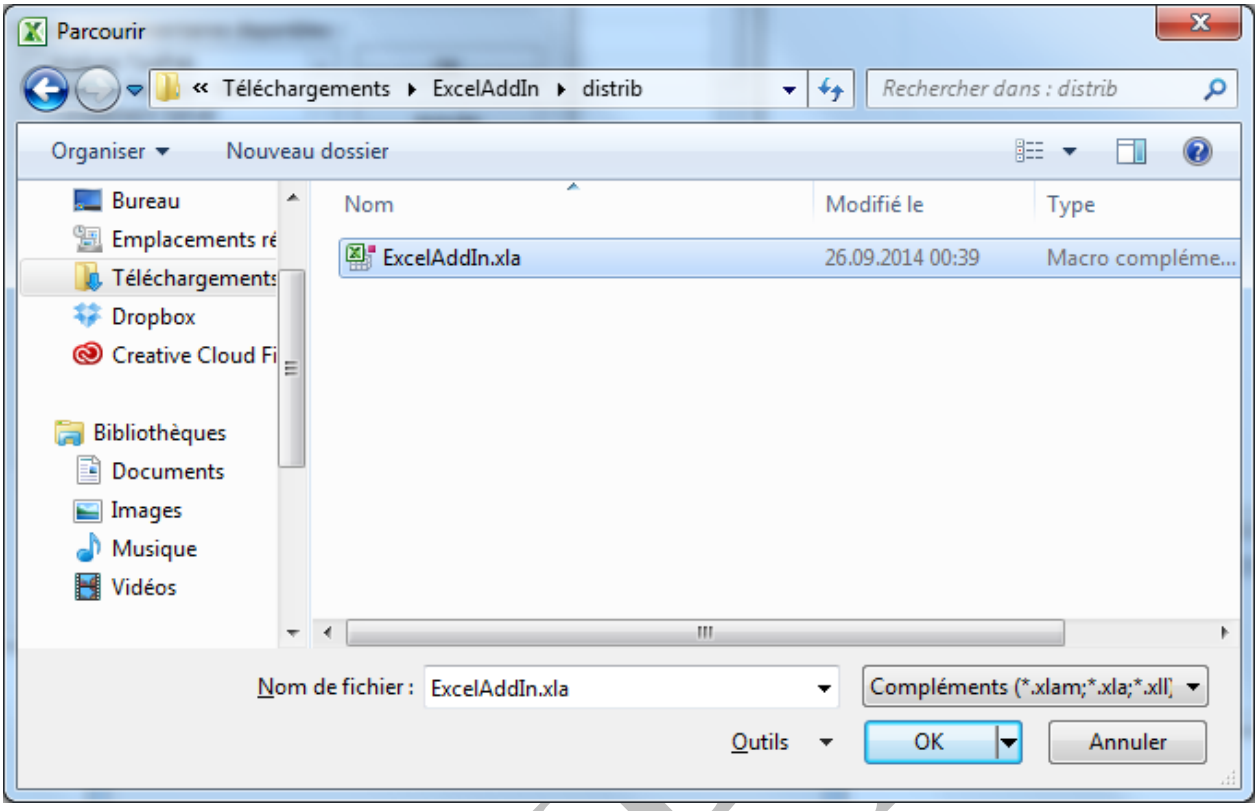
SA

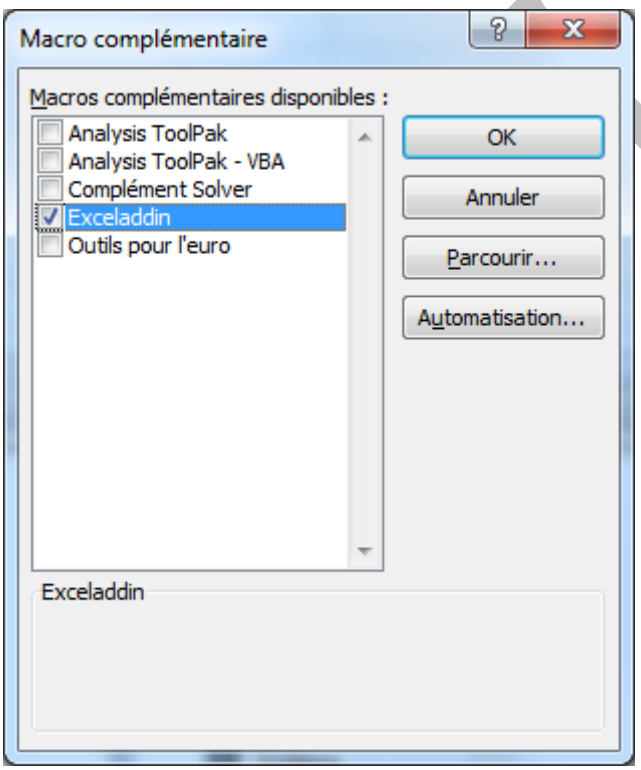

Isoz ▸ Téléchargements ▸ ExcelAddIn ▸ src
Rechercher dans : src
Fichier Edition Affichage Outils ?
Organiser Inclure dans la bibliothèque Partager avec Graver Nouveau dossier
Favoris
Bureau
Emplacements récents
Téléchargements
Dropbox
Creative Cloud Files
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
OS (C:)
iPhone Isoz Vincent
Réseau
Nom
Modifié le
Type
dlldata.c 26.09.2014 00:40 C Source
ExcelAddIn.bas 26.09.2014 00:39 Fichier BAS
ExcelAddIn.def 26.09.2014 00:39 Export Definition F...
ExcelAddIn.rc 26.09.2014 00:39 Fichier RC
ExcelAddIn.xla 26.09.2014 00:39 Macro compléme...
ExcelAddIn_1_0.dll 26.09.2014 00:40 Extension de l'app...
ExcelAddIn_1_0.exp 26.09.2014 00:40 Source Browser D...
ExcelAddIn_1_0.lib 26.09.2014 00:40 Source Browser D...
ExcelAddIn_dll.cpp 26.09.2014 00:39 C++ Source
ExcelAddIn_idl.h 26.09.2014 00:39 C/C++ Header
ExcelAddIn_idl.idl 26.09.2014 00:39 Interface Definitio...
ExcelAddIn_idl.tlb 26.09.2014 00:39 Fichier TLB
ExcelAddIn_idl_i.c 26.09.2014 00:39 C Source
ExcelAddIn_idl_p.c 26.09.2014 00:39 C Source
mccExcludedFiles.log 26.09.2014 00:39 Document texte
mwcomtypes.h 26.09.2014 00:40 C/C++ Header
mwcomtypes_i.c 26.09.2014 00:40 C Source
mwcomtypes_p.c 26.09.2014 00:40 C Source
mysum_com.cpp 26.09.2014 00:39 C++ Source
mysum_com.hpp 26.09.2014 00:39 C/C++ Header
readme.txt 26.09.2014 00:39 Document texte
21 élément(s)

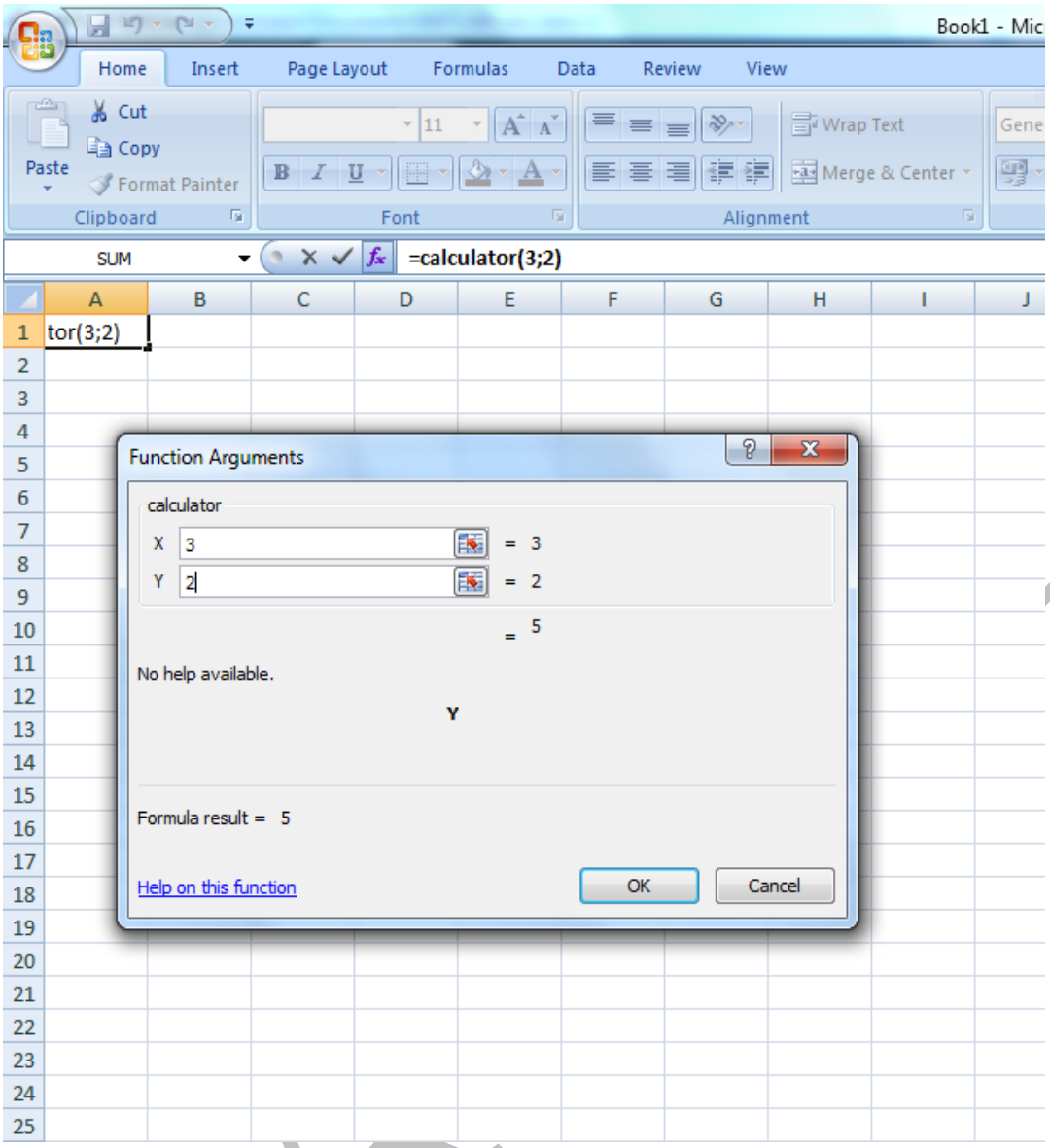
SA

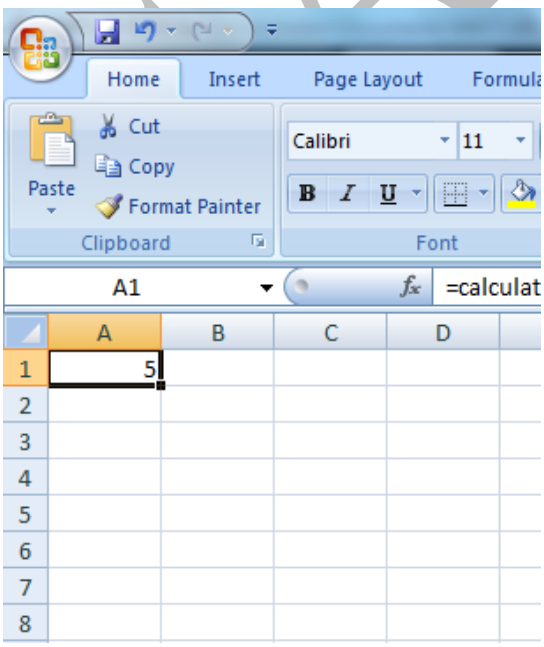

Options Excel
Général
Formules
Vérification
Enregistrement
Langue
Options avancées
Personnaliser le Ruban
Barre d'outils Accès rapide
Compléments
Centre de gestion de la confidentialité
Affichez et gérez les compléments Microsoft Office.
Compléments
Nom
Emplacement
Type
Compléments d'applications actifs
Acrobat PDFMaker Office COM Addin C:\...ice\PDFMOfficeAddin.dll Complément COM
Exceladdin C:\...In\distrib\ExcelAddIn.xla Complément Excel
Team Foundation Add-in "C:\...86\TFSOfficeAdd-in.dll" Complément COM
Compléments d'applications inactifs
Analysis ToolPak C:\...ry\Analysis\ANALYS32.XLL Complément Excel
Analysis ToolPak - VBA C:\...nalysis\ATPVBAEN.XLAM Complément Excel
Complément Solver C:\...ry\SOLVER\SOLVER.XLAM Complément Excel
Contenu invisible C:\...ce\Office14\OFFRHD.DLL Inspecteur de document
Données XML personnalisées C:\...ce\Office14\OFFRHD.DLL Inspecteur de document
En-têtes et pieds de page C:\...ce\Office14\OFFRHD.DLL Inspecteur de document
Feuilles de calcul cachées C:\...ce\Office14\OFFRHD.DLL Inspecteur de document
Lignes et colonnes cachées C:\...ce\Office14\OFFRHD.DLL Inspecteur de document
Microsoft Actions Pane 3 Kit d'extension XML
Outils pour l'euro C:\...Library\EUROTOOL.XLAM Complément Excel
Compléments liés au document
Aucun complément lié à un document
Compléments d'applications désactivés
Aucun complément d'application désactivé
Complément : Acrobat PDFMaker Office COM Addin
Éditeur : Adobe Systems, Incorporated
Compatibilité : Aucune information de compatibilité disponible
Emplacement : C:\Program Files (x86)\Adobe\Acrobat 9.0\PDFMaker\Office\PDFMOfficeAddin.dll
Description : Acrobat PDFMaker Office COM Addin
Gérer : Compléments Excel
Atteindre...
OK
Annuler


Macro complémentaire
Macros complémentaires disponibles :
Analysis ToolPak
Analysis ToolPak - VBA
Complément Solver
Outils pour l'euro
OK
Annuler
Parcourir...
Automatisation...
Exceladdin


Parcourir
« Téléchargements ▸ ExcelAddIn ▸ distrib
Rechercher dans : distrib
Organiser
Nouveau dossier
Bureau
Emplacements ré
Téléchargements
Dropbox
Creative Cloud Fi
Bibliothèques
Documents
Images
Musique
Vidéos
Nom
Modifié le
Type
ExcelAddIn.xla
26.09.2014 00:39
Macro compléme...
Nom de fichier :
ExcelAddIn.xla
Compléments (*.xlam;*.xla;*.xll
Outils
OK
Annuler


Macro complémentaire
Macros complémentaires disponibles :
Analysis ToolPak
Analysis ToolPak - VBA
Complément Solver
Exceladdin
Outils pour l'euro
OK
Annuler
Parcourir...
Automatisation...
Exceladdin

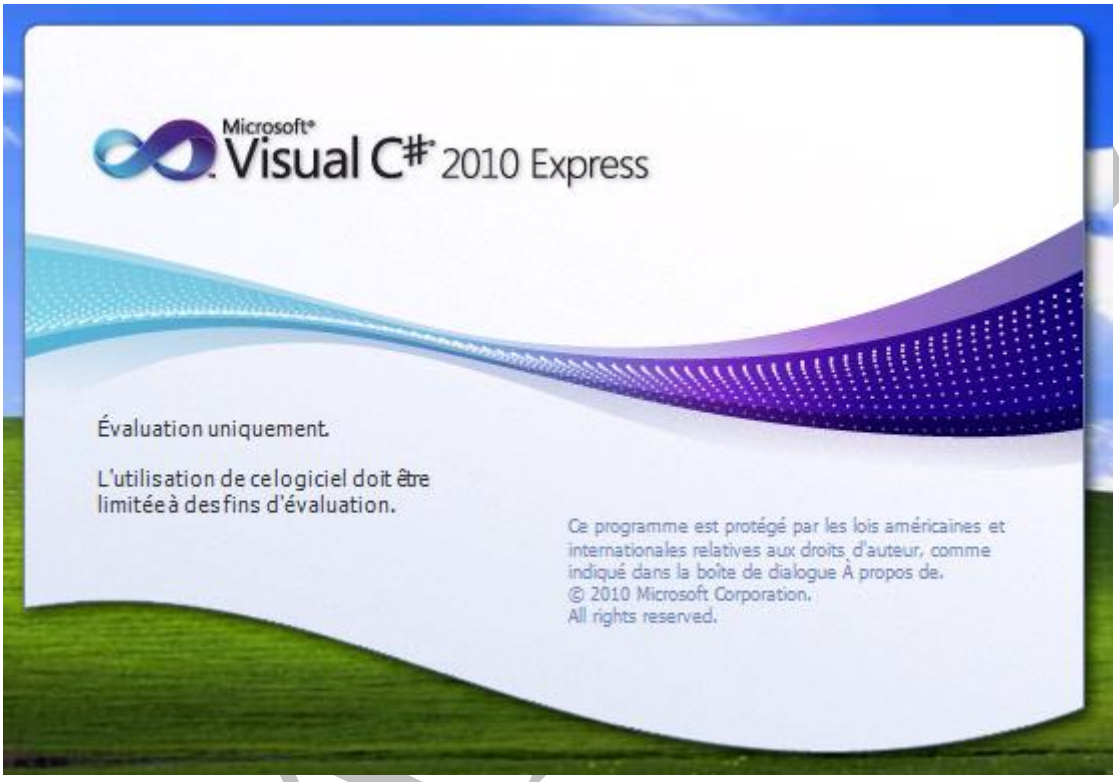

Book1 - Mic
Home
Insert
Page Layout
Formulas
Data
Review
View
Cut
Copy
Format Painter
Paste
Clipboard
Font
Alignment
Wrap Text
Merge & Center
SUM
=calculator(3;2)
tor(3;2)
Function Arguments
calculator
X 3 = 3
Y 2 = 2
= 5
No help available.
Y
Formula result = 5
Help on this function
OK
Cancel


Home
Insert
Page Layout
Cut
Copy
Format Painter
Paste
Clipboard
Font
Calibri
11
A1
=calculat
5

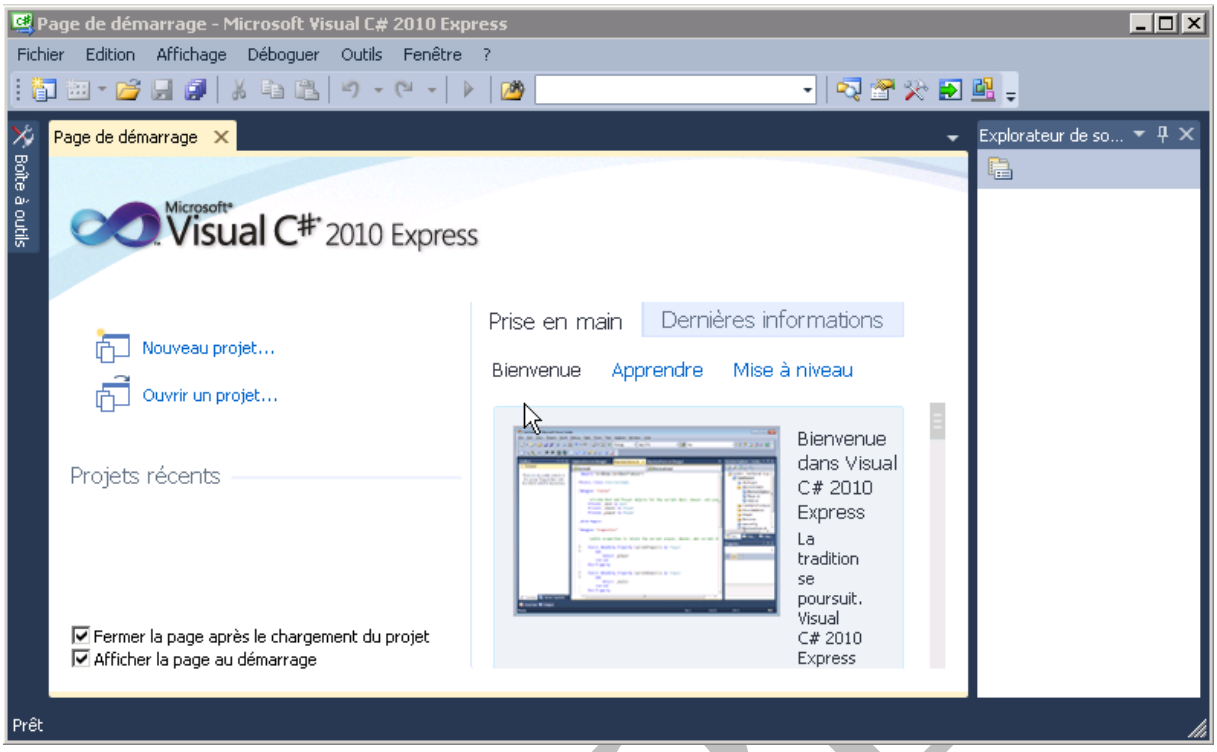
Use the *.dll and *.xla on another computer

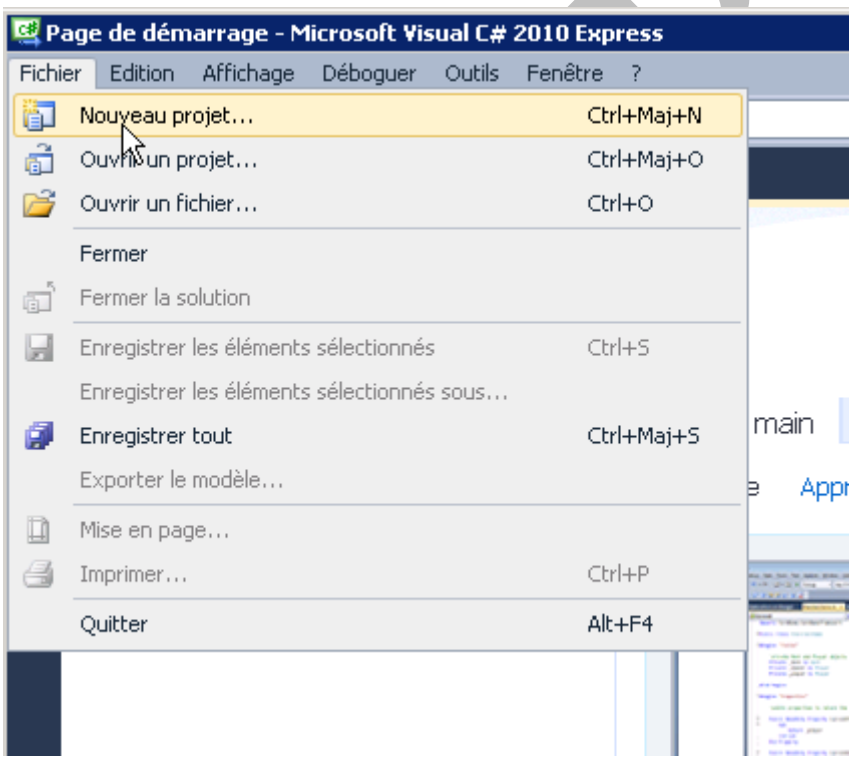
Call MATLAB from C# (VS Express 2010)

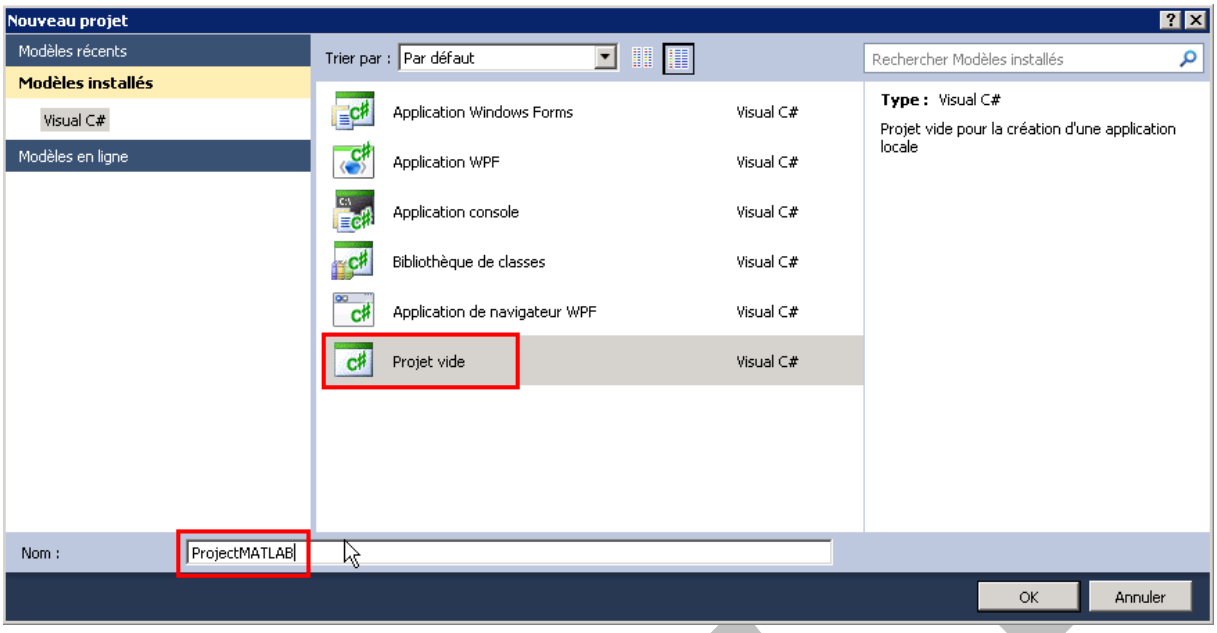
This part doesn't need any toolboxes:

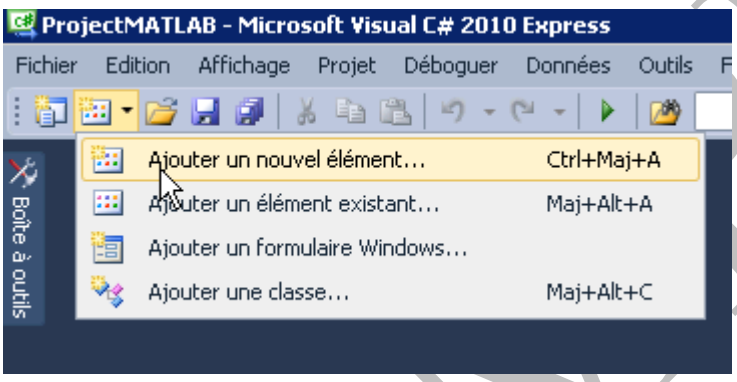


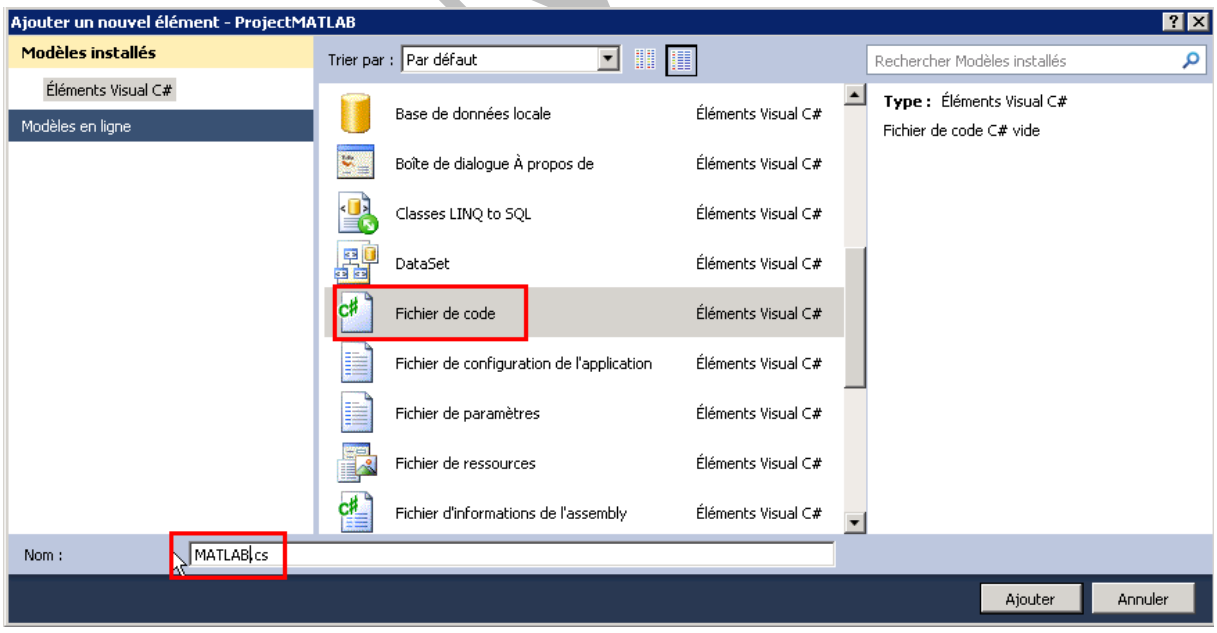
SA

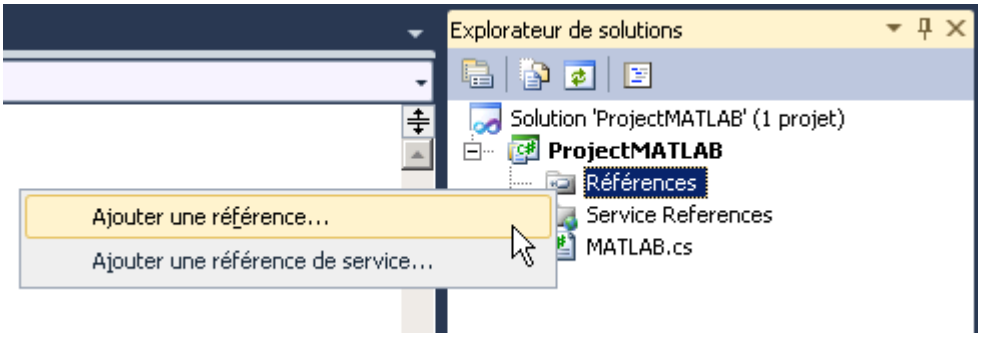

Page de démarrage - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Déboguer Outils Fenêtre ?
Page de démarrage
Boîte à outils
Explorateur de so...
Microsoft Visual C# 2010 Express
Nouveau projet...
Ouvrir un projet...
Projets récents
Prise en main
Dernières informations
Bienvenue
Apprendre
Mise à niveau
Bienvenue dans Visual C# 2010 Express
La tradition se poursuit. Visual C# 2010 Express
Fermer la page après le chargement du projet
Afficher la page au démarrage
Prêt

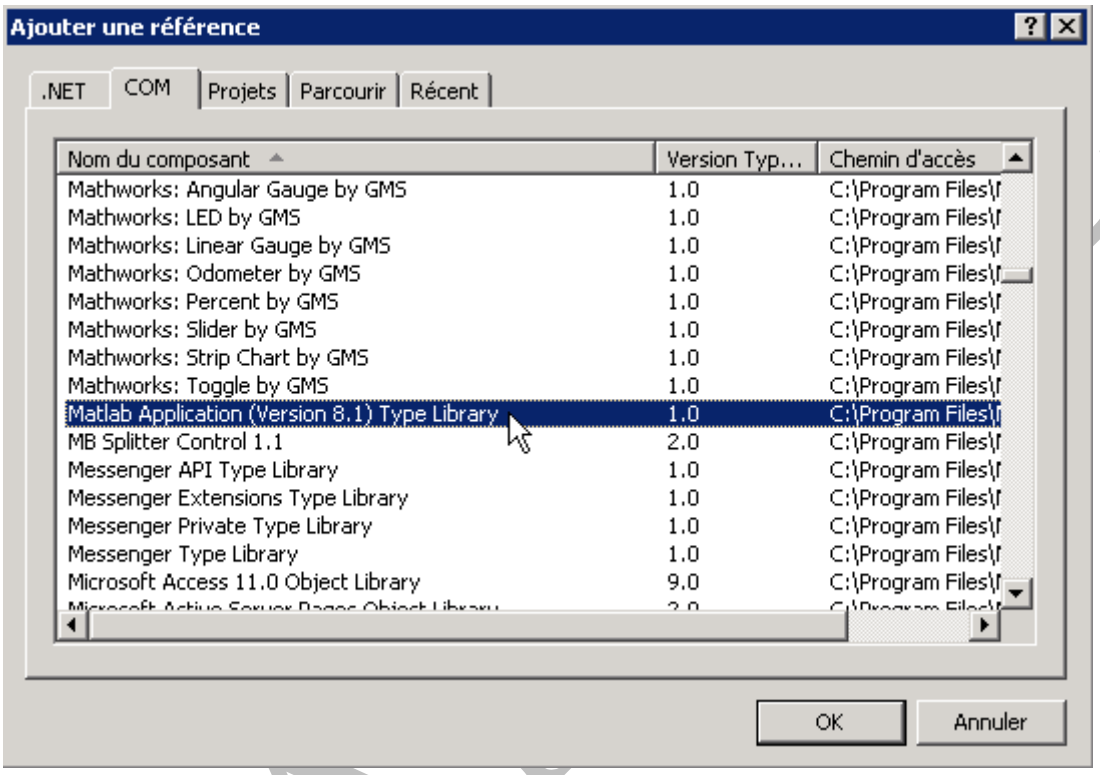

Page de démarrage - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Déboguer Outils Fenêtre ?
Nouveau projet... Ctrl+Maj+N
Ouvrir un projet... Ctrl+Maj+O
Ouvrir un fichier... Ctrl+O
Fermer
Fermer la solution
Enregistrer les éléments sélectionnés Ctrl+S
Enregistrer les éléments sélectionnés sous...
Enregistrer tout Ctrl+Maj+S
Exporter le modèle...
Mise en page...
Imprimer... Ctrl+P
Quitter Alt+F4
main


Nouveau projet
Modèles récents
Modèles installés
Visual C#
Modèles en ligne
Trier par : Par défaut
Rechercher Modèles installés
Application Windows Forms Visual C#
Application WPF Visual C#
Application console Visual C#
Bibliothèque de classes Visual C#
Application de navigateur WPF Visual C#
Projet vide Visual C#
Type : Visual C#
Projet vide pour la création d'une application locale
Nom : ProjectMATLAB
OK
Annuler


ProjectMATLAB - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données Outils
Ajouter un nouvel élément... Ctrl+Maj+A
Ajouter un élément existant... Maj+Alt+A
Ajouter un formulaire Windows...
Ajouter une classe... Maj+Alt+C
Boîte à outils

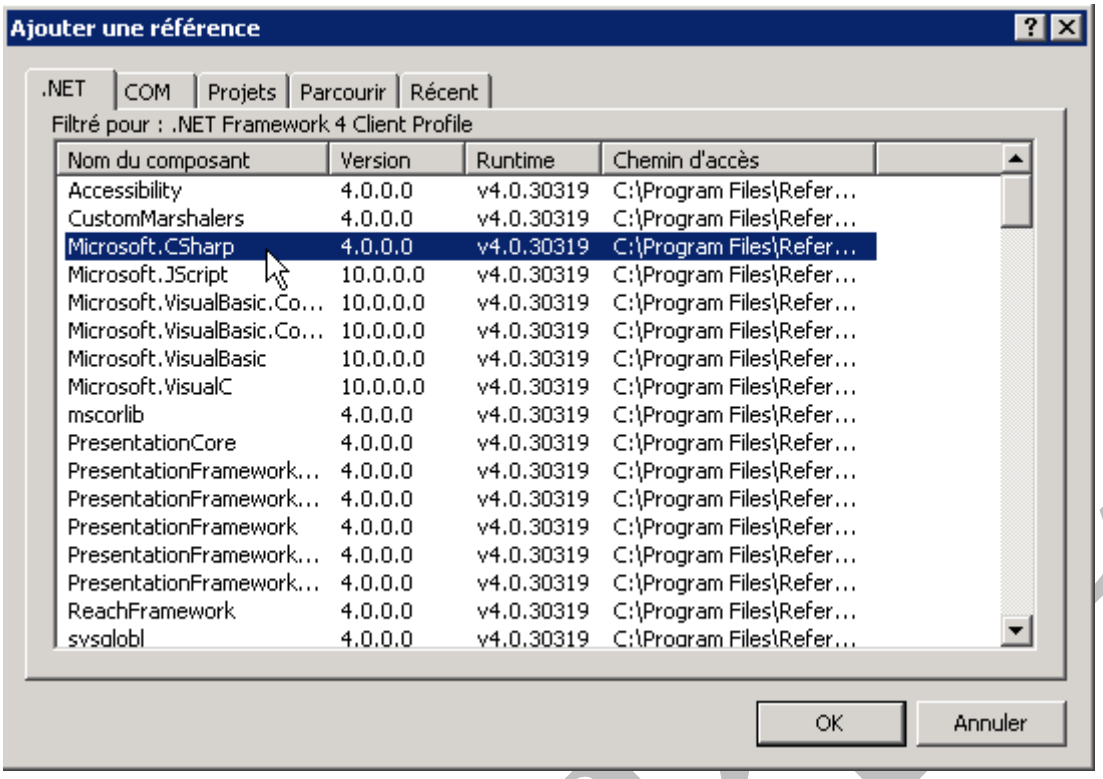

Ajouter un nouvel élément - ProjectMATLAB
Modèles installés
Éléments Visual C#
Modèles en ligne
Trier par : Par défaut
Rechercher Modèles installés
Base de données locale Éléments Visual C#
Boîte de dialogue À propos de Éléments Visual C#
Classes LINQ to SQL Éléments Visual C#
DataSet Éléments Visual C#
Fichier de code Éléments Visual C#
Fichier de configuration de l'application Éléments Visual C#
Fichier de paramètres Éléments Visual C#
Fichier de ressources Éléments Visual C#
Fichier d'informations de l'assembly Éléments Visual C#
Type : Éléments Visual C#
Fichier de code C# vide
Nom : MATLAB.cs
Ajouter
Annuler

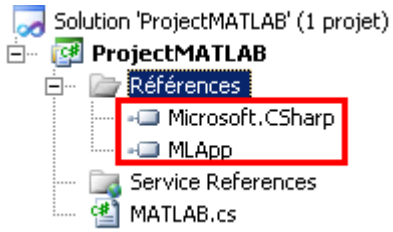




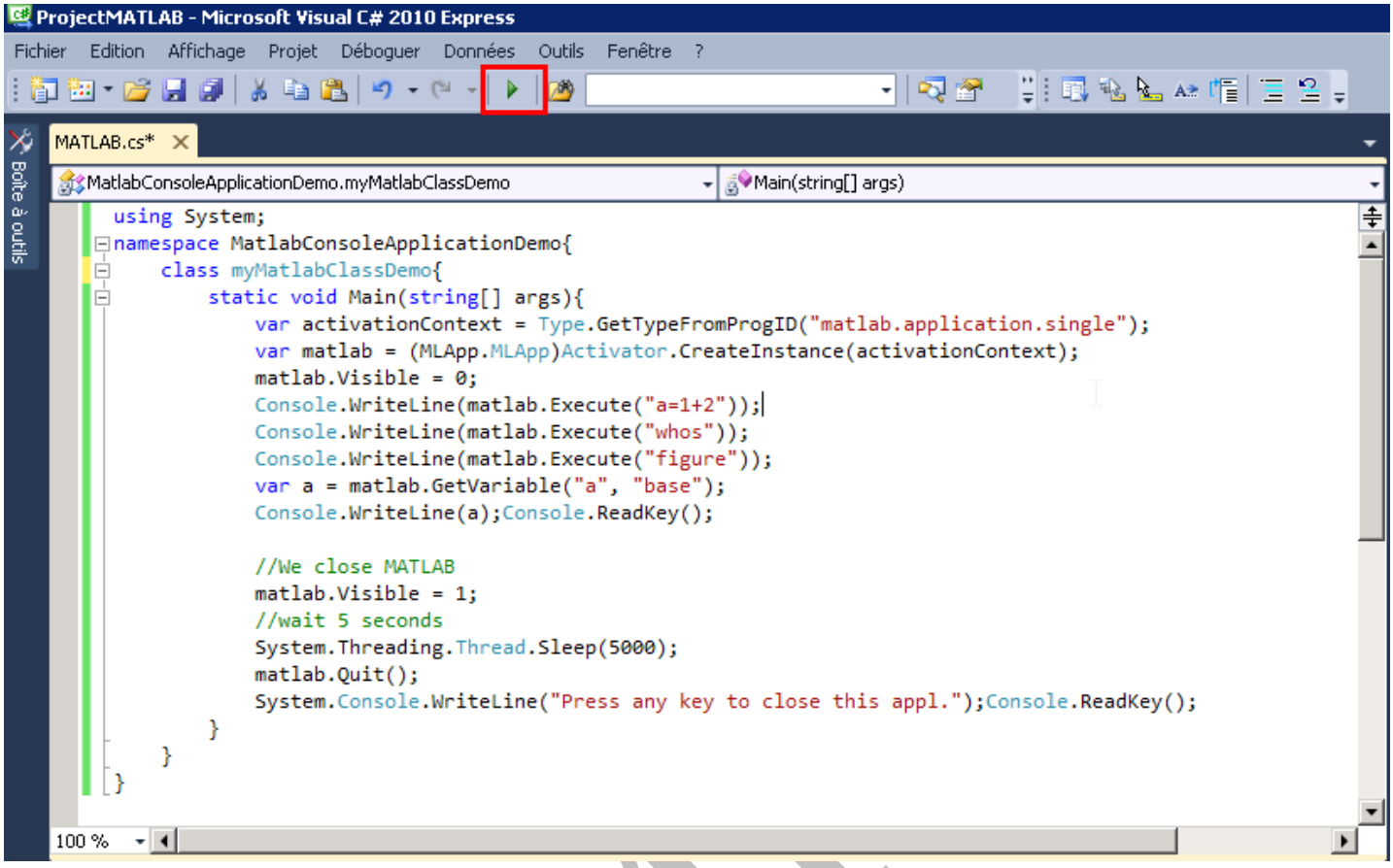
COM technology cons: MATLAB is required to be installed on the target computer.

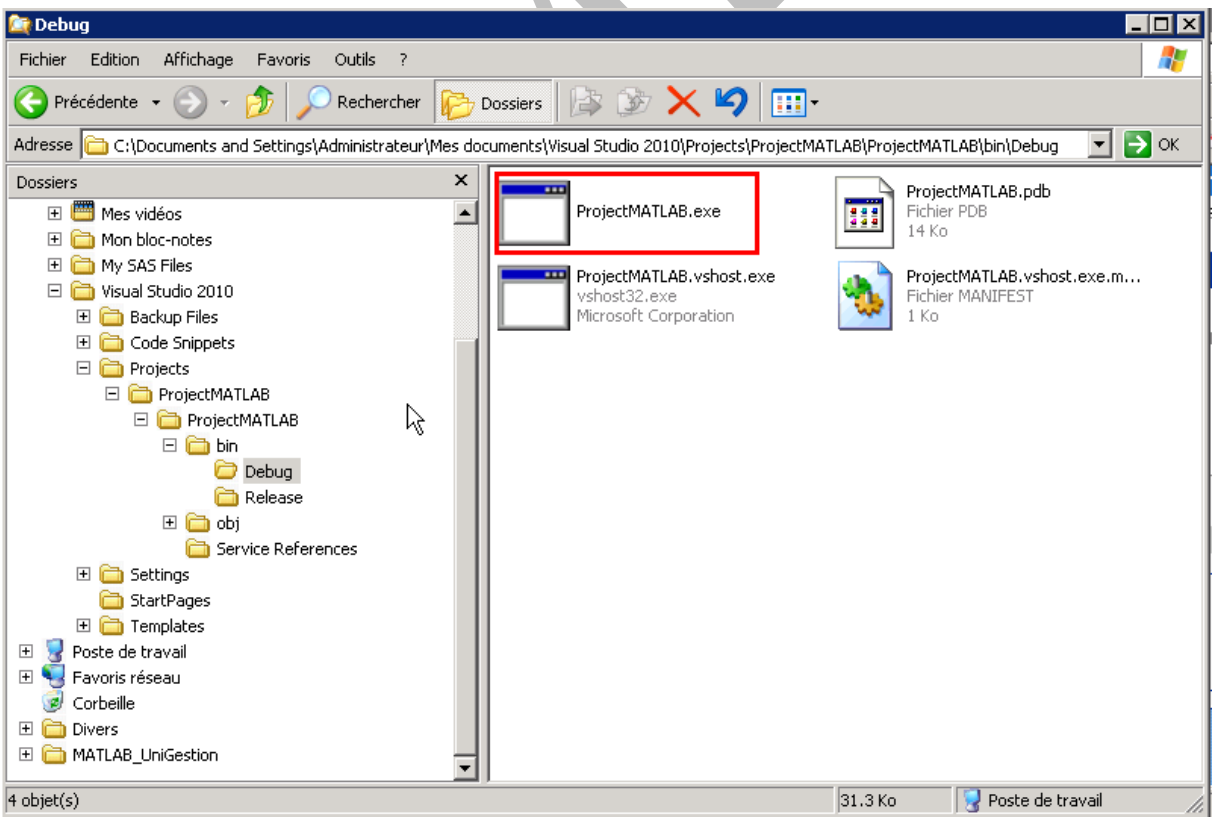
MATLAB .NET builder compiles your MATLAB code to the .NET assembly and you can use it directly.

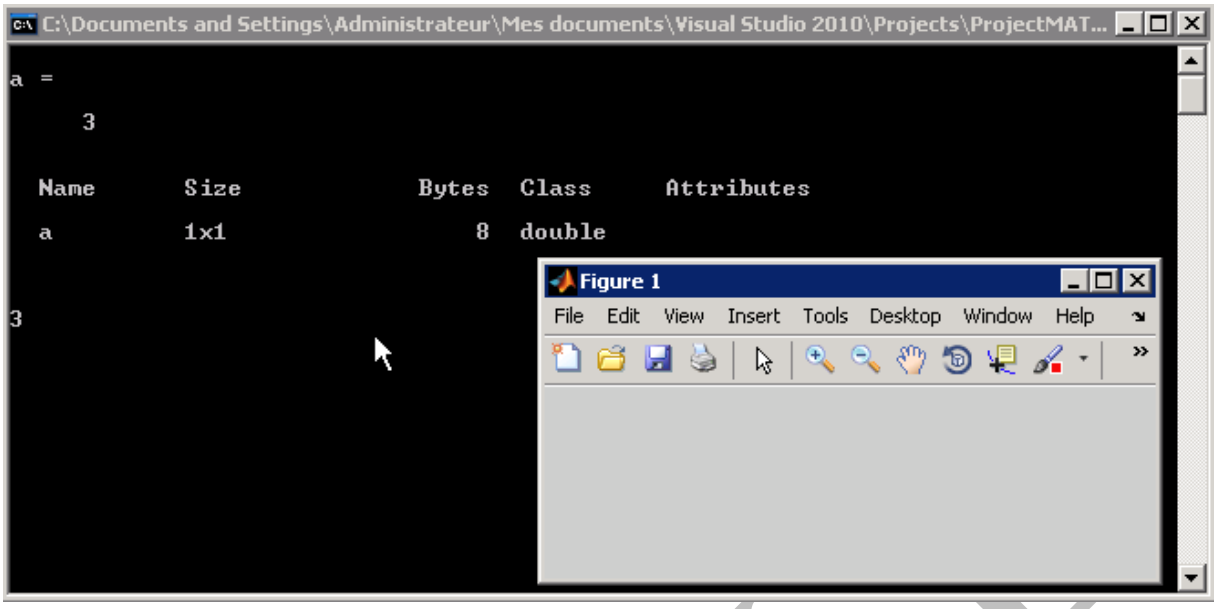

Ajouter une référence
.NET
COM
Projets
Parcourir
Récent
Filtré pour : .NET Framework 4 Client Profile
Nom du composant
Version
Runtime
Chemin d'accès
Accessibility 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
CustomMarshalers 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.CSharp 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.JScript 10.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.VisualBasic.Co... 10.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.VisualBasic.Co... 10.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.VisualBasic 10.0.0.0 v4.0.30319 C:\Program Files\Refer...
Microsoft.VisualC 10.0.0.0 v4.0.30319 C:\Program Files\Refer...
mscorlib 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationCore 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationFramework... 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationFramework... 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationFramework 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationFramework... 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
PresentationFramework... 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
ReachFramework 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
sysglobl 4.0.0.0 v4.0.30319 C:\Program Files\Refer...
OK
Annuler

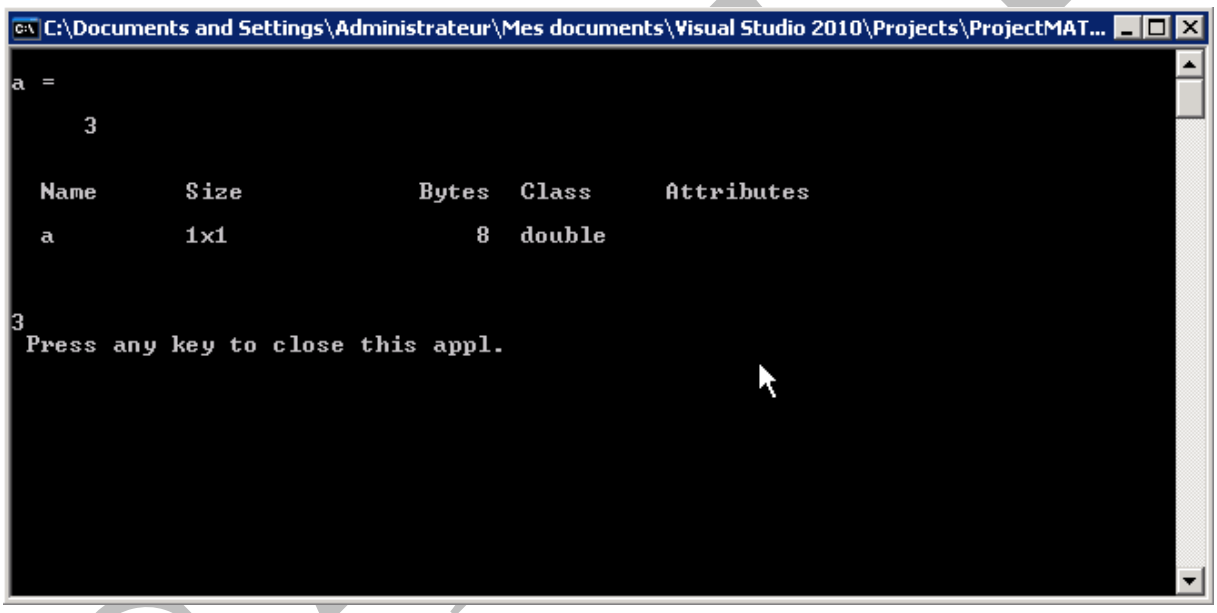

Solution 'ProjectMATLAB' (1 projet)
ProjectMATLAB
Références
Microsoft.CSharp
MLApp
Service References
MATLAB.cs

SAMPLE


ProjectMATLAB - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données Outils Fenêtre ?
MATLAB.cs*
MatlabConsoleApplicationDemo.myMatlabClassDemo
Main(string[] args)
using System;
namespace MatlabConsoleApplicationDemo{
class myMatlabClassDemo{
static void Main(string[] args){
var activationContext = Type.GetTypeFromProgID("matlab.application.single");
var matlab = (MLApp.MLApp)Activator.CreateInstance(activationContext);
matlab.Visible = 0;
Console.WriteLine(matlab.Execute("a=1+2"));
Console.WriteLine(matlab.Execute("whos"));
Console.WriteLine(matlab.Execute("figure"));
var a = matlab.GetVariable("a", "base");
Console.WriteLine(a);Console.ReadKey();
//We close MATLAB
matlab.Visible = 1;
//wait 5 seconds
System.Threading.Thread.Sleep(5000);
matlab.Quit();
System.Console.WriteLine("Press any key to close this appl.");Console.ReadKey();
}
}
}
100 %

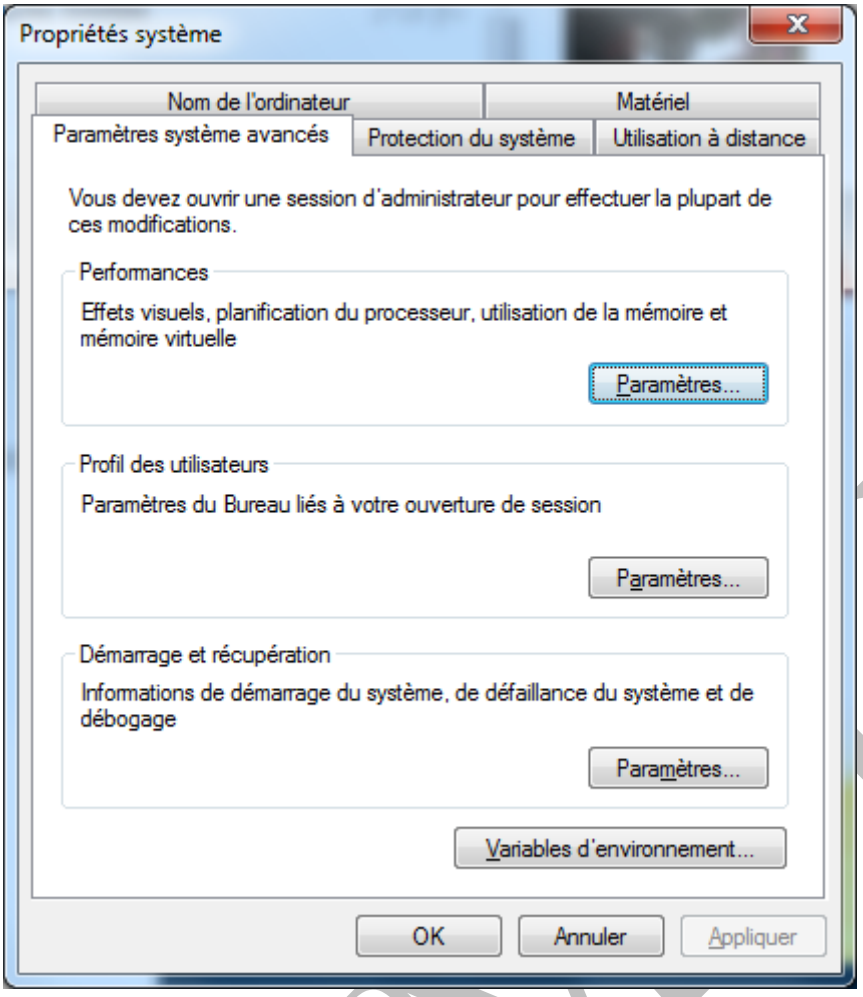

Debug
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Dossiers
Adresse C:\Documents and Settings\Administrateur\Mes documents\Visual Studio 2010\Projects\ProjectMATLAB\ProjectMATLAB\bin\Debug
OK
Dossiers
Mes vidéos
Mon bloc-notes
My SAS Files
Visual Studio 2010
Backup Files
Code Snippets
Projects
ProjectMATLAB
ProjectMATLAB
bin
Debug
Release
obj
Service References
Settings
StartPages
Templates
Poste de travail
Favoris réseau
Corbeille
Divers
MATLAB_UniGestion
ProjectMATLAB.exe
ProjectMATLAB.pdb
Fichier PDB
14 Ko
ProjectMATLAB.vshost.exe
vshost32.exe
Microsoft Corporation
ProjectMATLAB.vshost.exe.m...
Fichier MANIFEST
1 Ko
4 objet(s)
31.3 Ko
Poste de travail



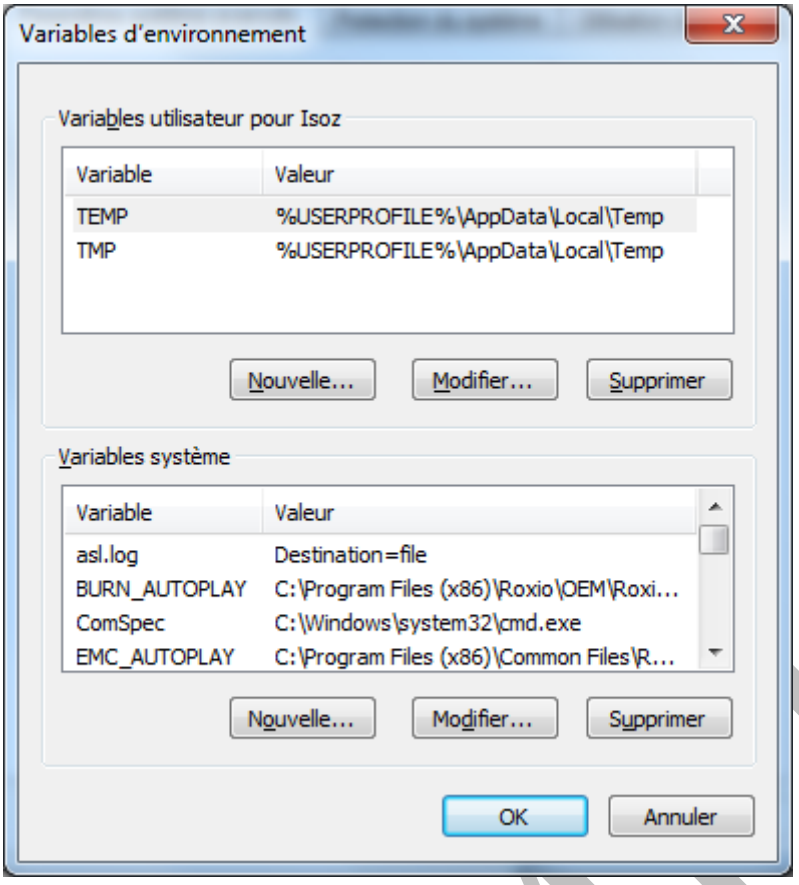


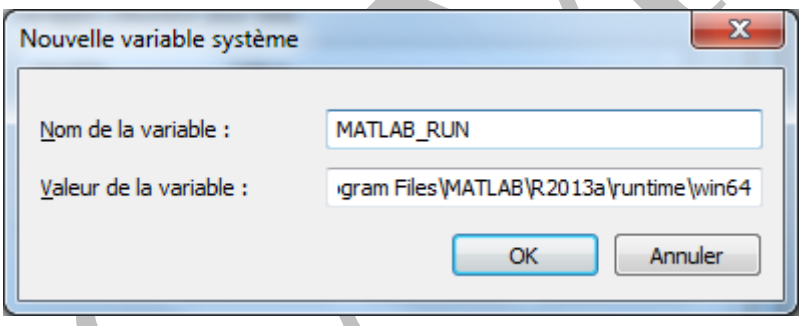
Call MATLAB from C++ (VS Express 2010 x64)

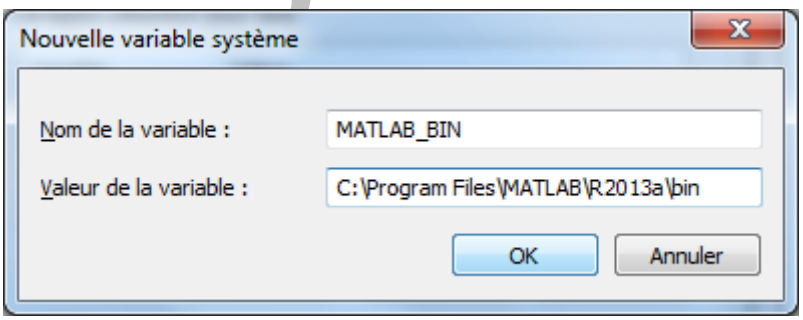
Some steps will seems to be duplicates when you will read what follows but in the facts it is just a cumuluted return on experience troubleshooting manipulations to avoid any problems :


Propriétés système
Nom de l'ordinateur
Matériel
Paramètres système avancés
Protection du système
Utilisation à distance
Vous devez ouvrir une session d'administrateur pour effectuer la plupart de ces modifications.
Performances
Effets visuels, planification du processeur, utilisation de la mémoire et mémoire virtuelle
Paramètres...
Profil des utilisateurs
Paramètres du Bureau liés à votre ouverture de session
Paramètres...
Démarrage et récupération
Informations de démarrage du système, de défaillance du système et de débogage
Paramètres...
Variables d'environnement...
OK
Annuler
Appliquer

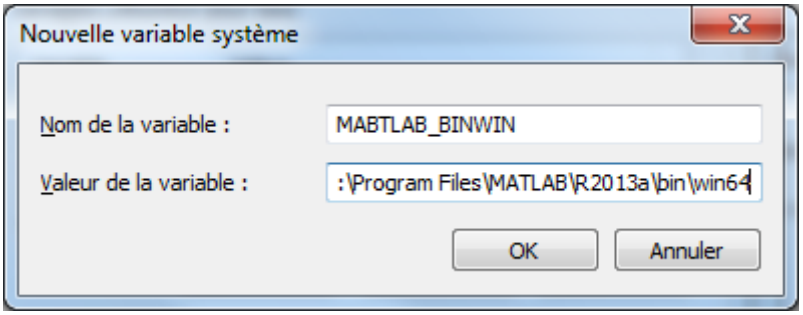
SAMPLE


Variables d'environnement
Variables utilisateur pour Isoz
Variable
Valeur
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
Nouvelle...
Modifier...
Supprimer
Variables système
Variable
Valeur
asl.log
Destination=file
BURN_AUTOPLAY
C:\Program Files (x86)\Roxio\OEM\Roxi...
ComSpec
C:\Windows\system32\cmd.exe
EMC_AUTOPLAY
C:\Program Files (x86)\Common Files\R...
Nouvelle...
Modifier...
Supprimer
OK
Annuler


Nouvelle variable système
Nom de la variable :
MATLAB_RUN
Valeur de la variable :
gram Files\MATLAB\R2013a\runtime\win64
OK
Annuler

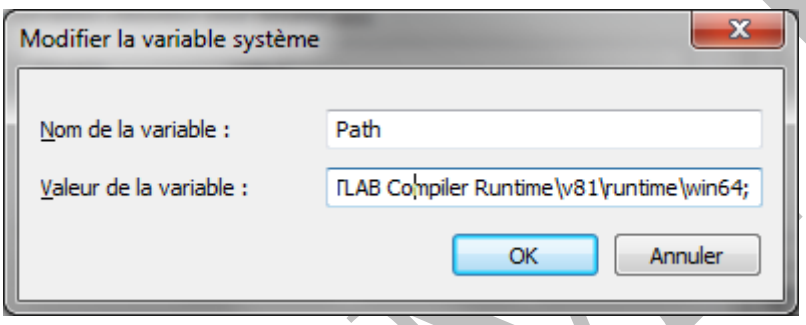

Nouvelle variable système
Nom de la variable :
MATLAB_BIN
Valeur de la variable :
C:\Program Files\MATLAB\R2013a\bin
OK
Annuler

SAMPLE



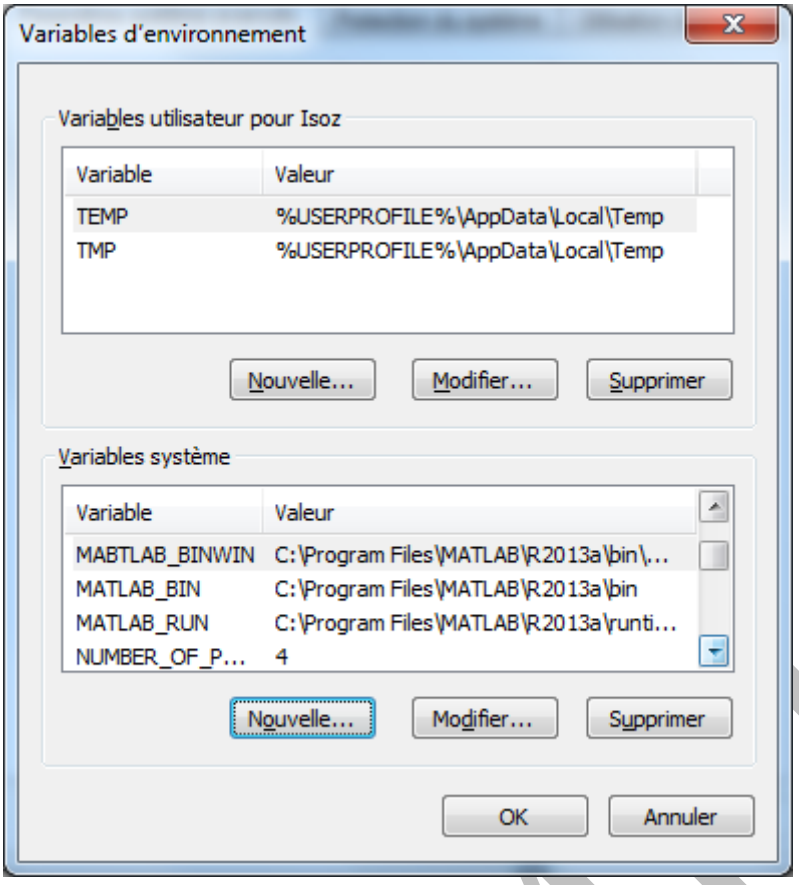
And check that in PATH you have anywhere :

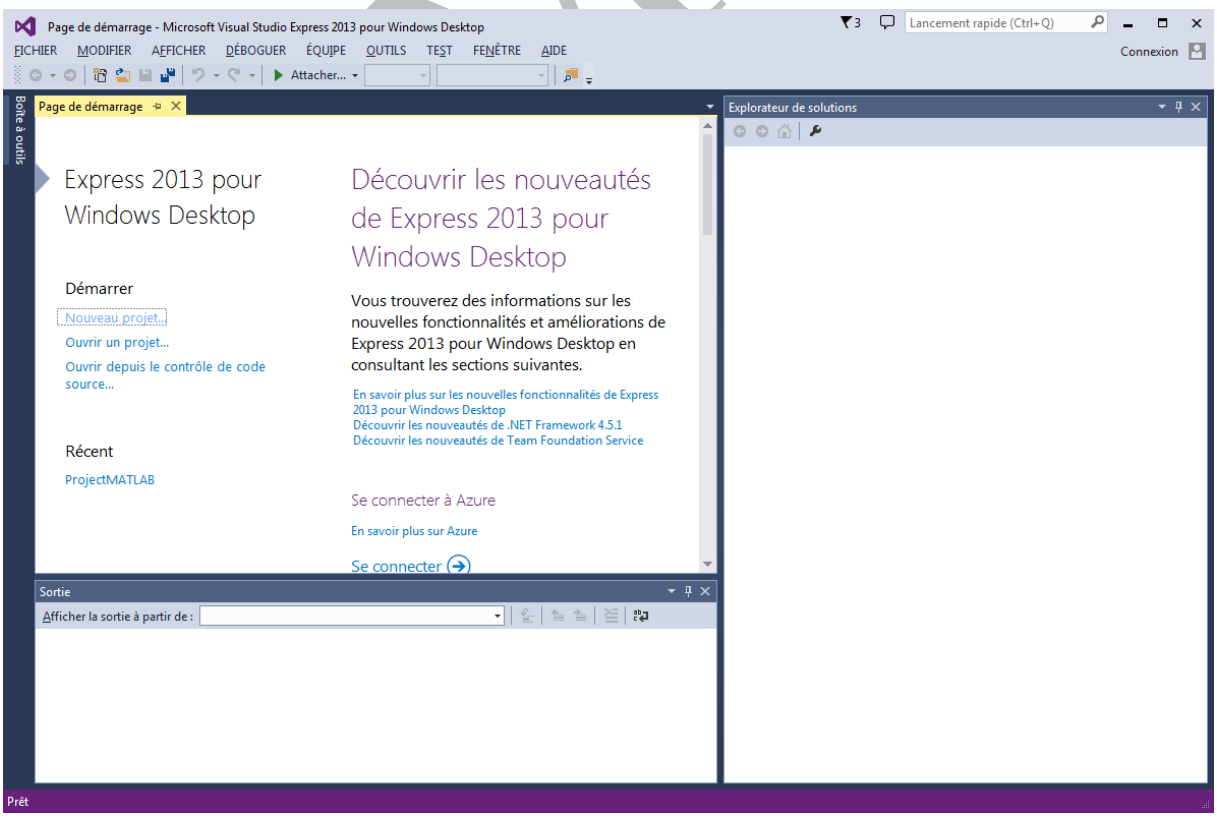
C:\Program Files\MATLAB\MATLAB Compiler Runtime\v81\bin\win64



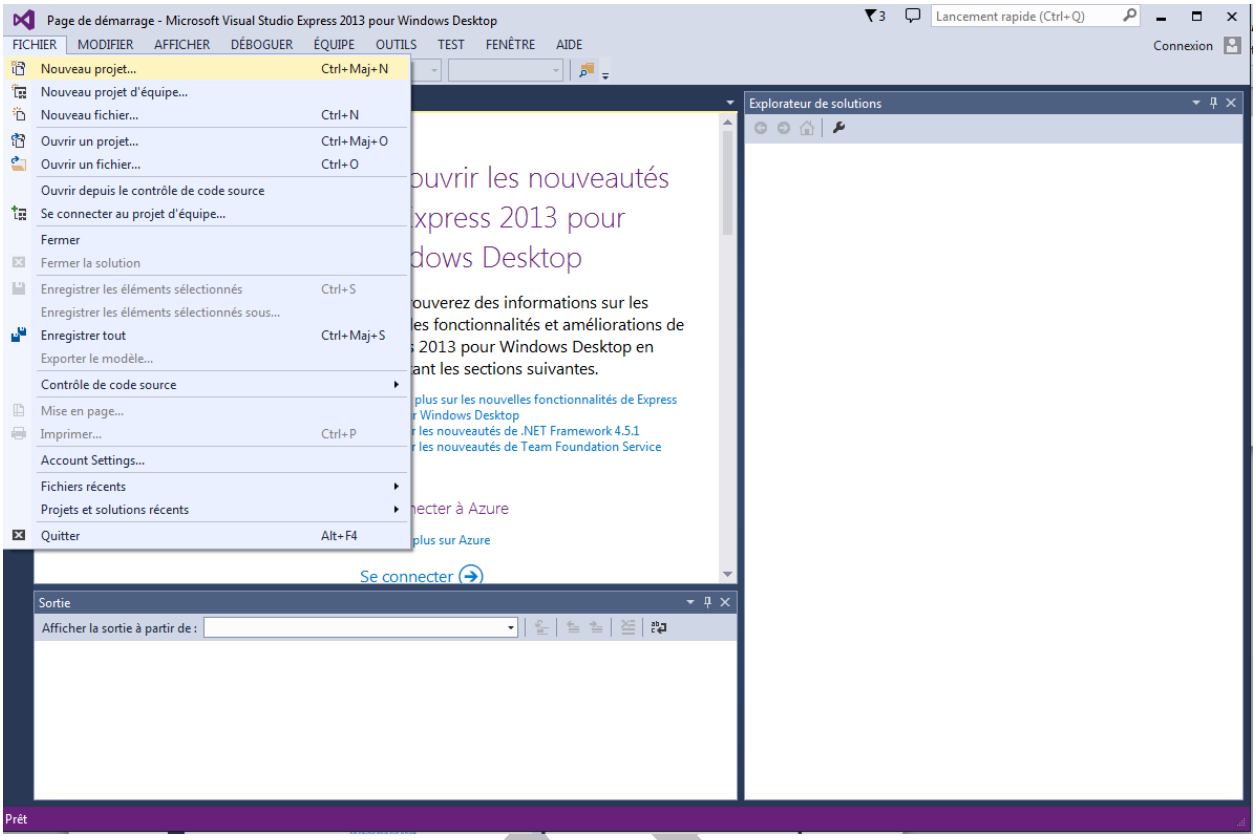
Restart Computer !

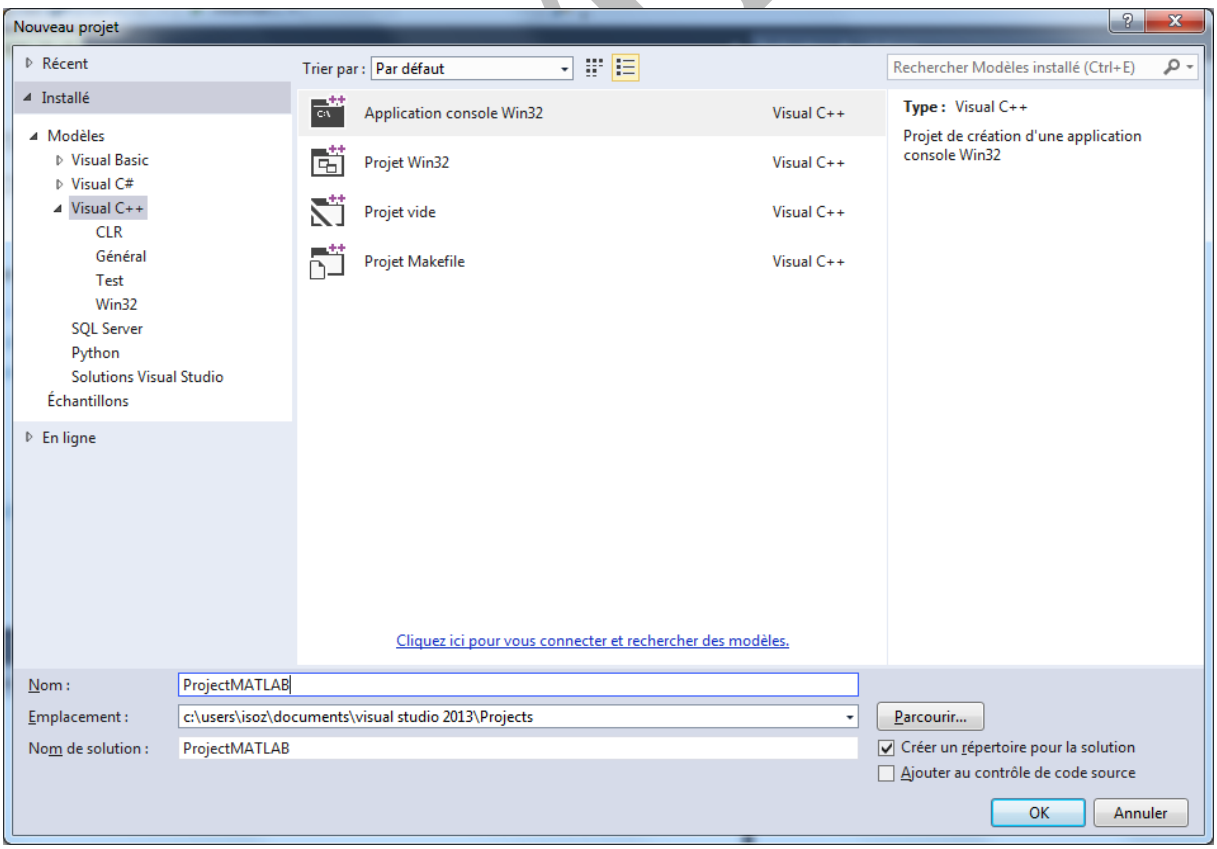
SAMPLE

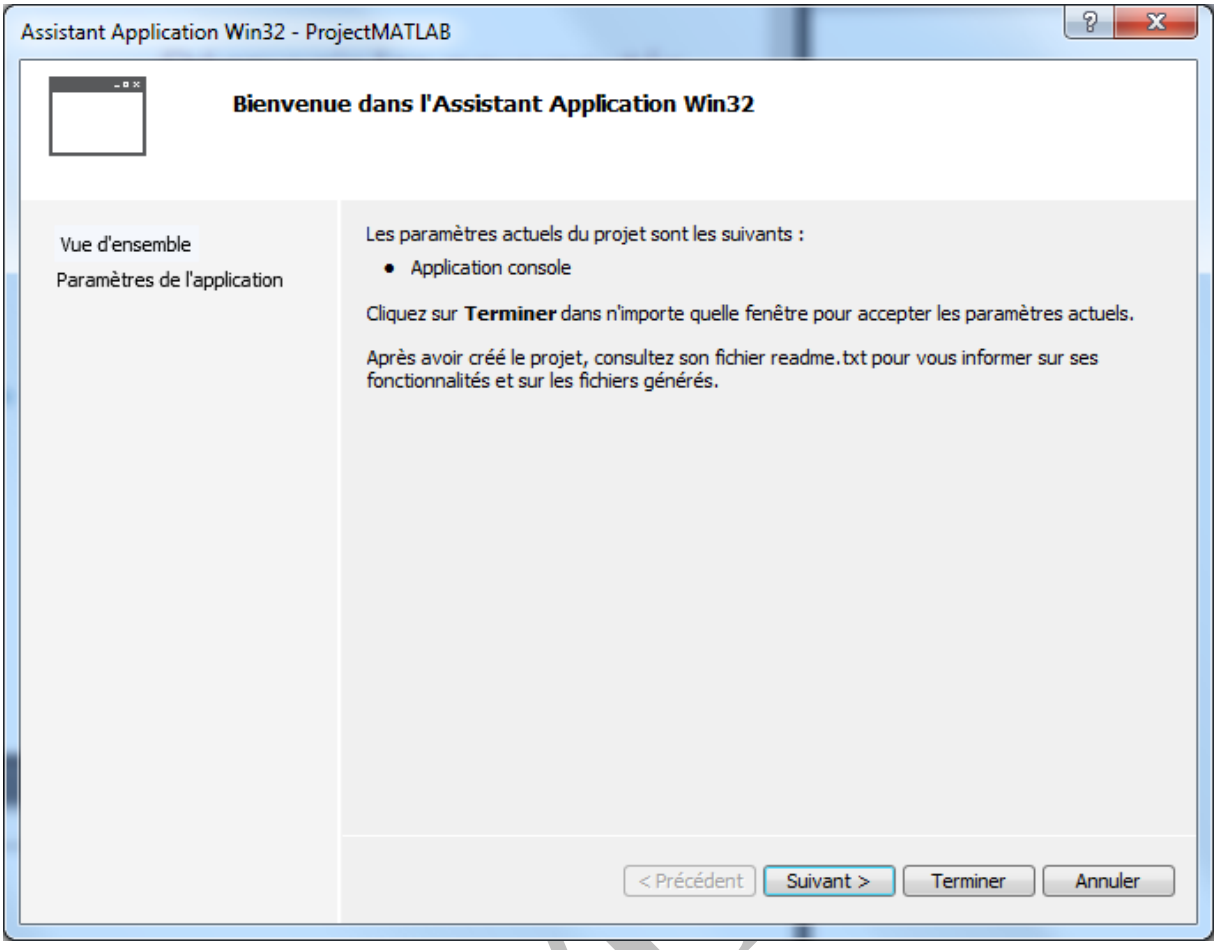

Variables d'environnement
Variables utilisateur pour Isoz
Variable
Valeur
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
Nouvelle...
Modifier...
Supprimer
Variables système
Variable
Valeur
MABTLAB_BINWIN
C:\Program Files\MATLAB\R2013a\bin\...
MATLAB_BIN
C:\Program Files\MATLAB\R2013a\bin
MATLAB_RUN
C:\Program Files\MATLAB\R2013a\runti...
NUMBER_OF_P...
4
Nouvelle...
Modifier...
Supprimer
OK
Annuler


Page de démarrage - Microsoft Visual Studio Express 2013 pour Windows Desktop
FICHIER MODIFIER AFFICHER DÉBOGUER ÉQUIPE OUTILS TEST FENÊTRE AIDE
Lancement rapide (Ctrl+Q)
Connexion
Attacher...
Boîte à outils
Page de démarrage
Express 2013 pour Windows Desktop
Démarrer
Nouveau projet...
Ouvrir un projet...
Ouvrir depuis le contrôle de code source...
Récent
ProjectMATLAB
Découvrir les nouveautés de Express 2013 pour Windows Desktop
Vous trouverez des informations sur les nouvelles fonctionnalités et améliorations de Express 2013 pour Windows Desktop en consultant les sections suivantes.
En savoir plus sur les nouvelles fonctionnalités de Express 2013 pour Windows Desktop
Découvrir les nouveautés de .NET Framework 4.5.1
Découvrir les nouveautés de Team Foundation Service
Se connecter à Azure
En savoir plus sur Azure
Se connecter
Explorateur de solutions
Sortie
Afficher la sortie à partir de :
Prêt

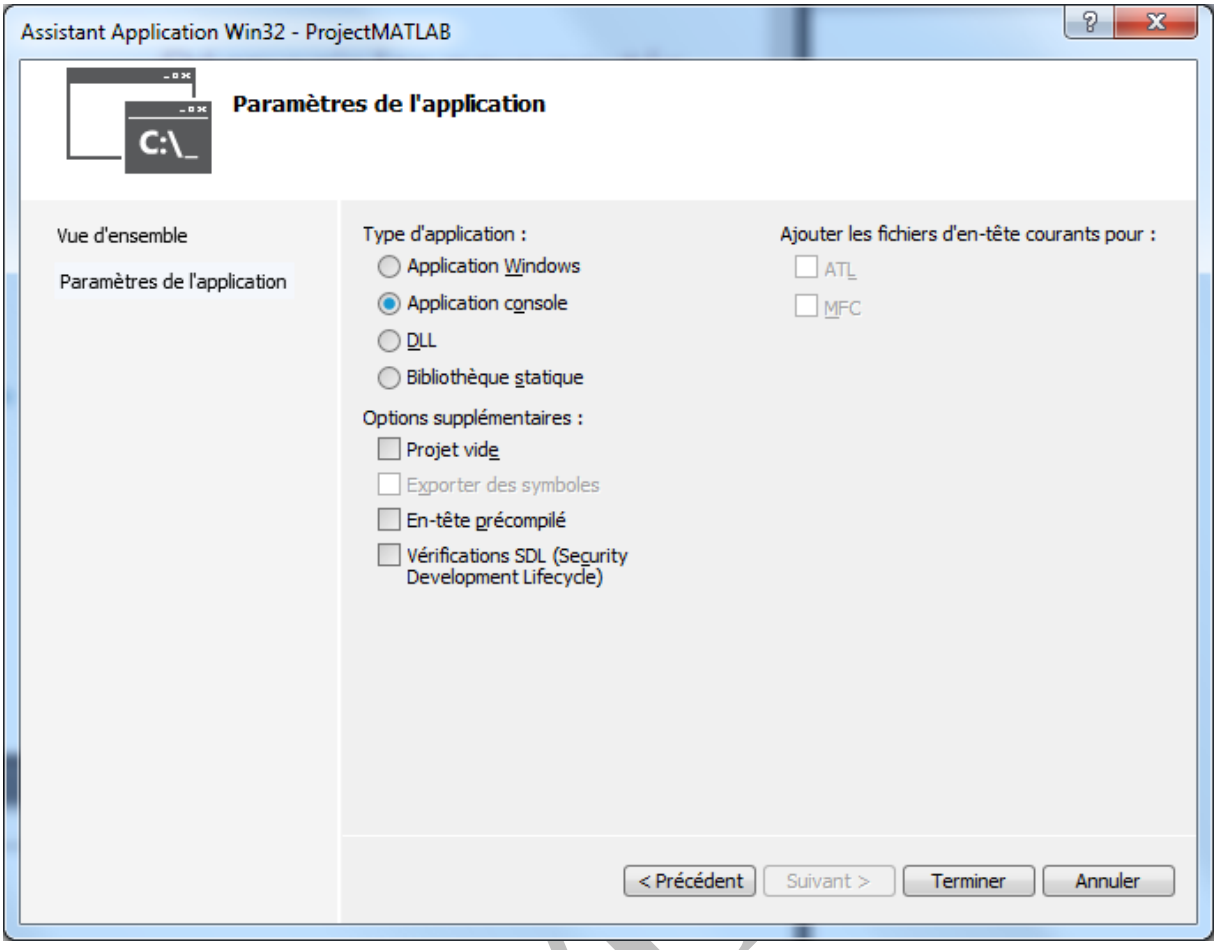
SAMPLE

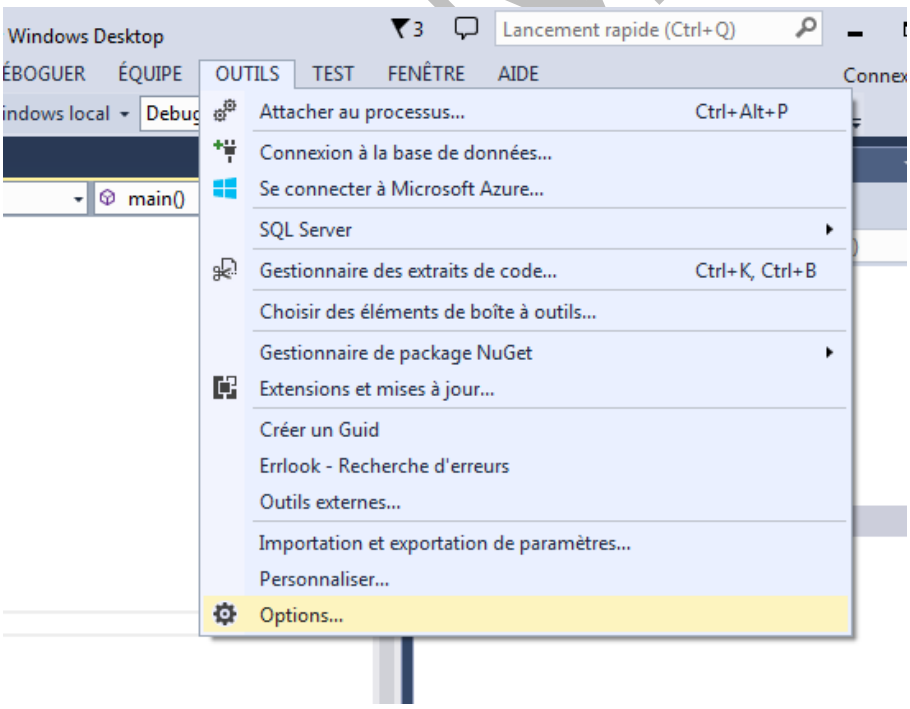

Page de démarrage - Microsoft Visual Studio Express 2013 pour Windows Desktop
FICHIER MODIFIER AFFICHER DÉBOGUER ÉQUIPE OUTILS TEST FENÊTRE AIDE
Nouveau projet... Ctrl+Maj+N
Nouveau projet d'équipe...
Nouveau fichier... Ctrl+N
Ouvrir un projet... Ctrl+Maj+O
Ouvrir un fichier... Ctrl+O
Ouvrir depuis le contrôle de code source
Se connecter au projet d'équipe...
Fermer
Fermer la solution
Enregistrer les éléments sélectionnés Ctrl+S
Enregistrer les éléments sélectionnés sous...
Enregistrer tout Ctrl+Maj+S
Exporter le modèle...
Contrôle de code source
Mise en page...
Imprimer... Ctrl+P
Account Settings...
Fichiers récents
Projets et solutions récents
Quitter Alt+F4
Lancement rapide (Ctrl+Q)
Connexion
Explorateur de solutions
Se connecter
Sortie
Afficher la sortie à partir de :
Prêt

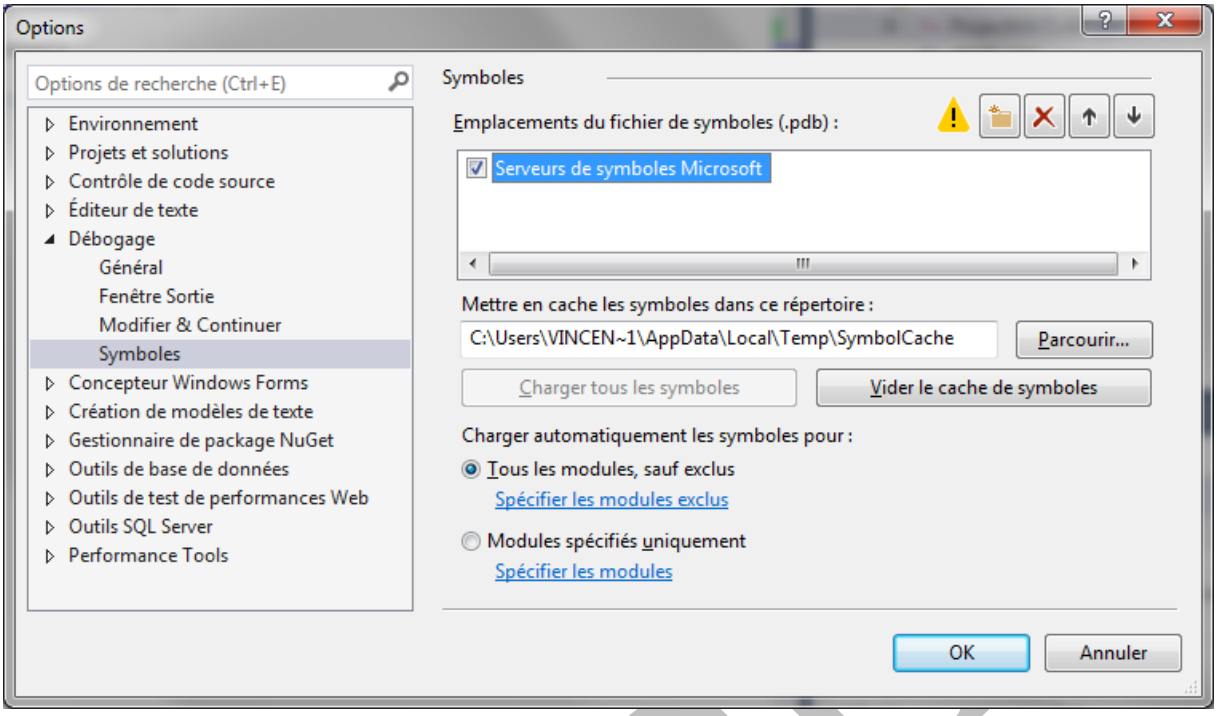

Nouveau projet
Récent
Installé
Modèles
Visual Basic
Visual C#
Visual C++
CLR
Général
Test
Win32
SQL Server
Python
Solutions Visual Studio
Échantillons
En ligne
Trier par : Par défaut
Application console Win32 Visual C++
Projet Win32 Visual C++
Projet vide Visual C++
Projet Makefile Visual C++
Rechercher Modèles installé (Ctrl+E)
Type : Visual C++
Projet de création d'une application console Win32
Cliquez ici pour vous connecter et rechercher des modèles.
Nom : ProjectMATLAB
Emplacement : c:\users\isoz\documents\visual studio 2013\Projects
Parcourir...
Nom de solution : ProjectMATLAB
Créer un répertoire pour la solution
Ajouter au contrôle de code source
OK
Annuler

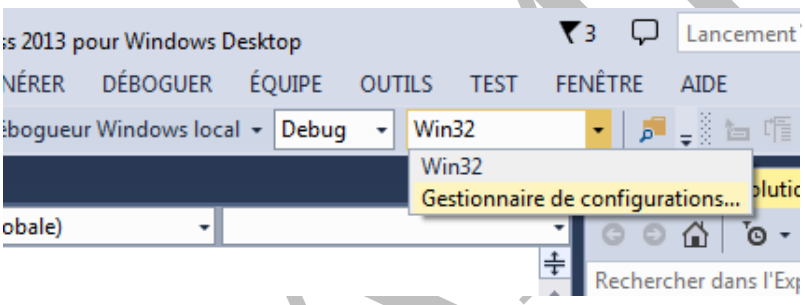

Assistant Application Win32 - ProjectMATLAB
Bienvenue dans l'Assistant Application Win32
Vue d'ensemble
Paramètres de l'application
Les paramètres actuels du projet sont les suivants :
Application console
Cliquez sur Terminer dans n'importe quelle fenêtre pour accepter les paramètres actuels.
Après avoir créé le projet, consultez son fichier readme.txt pour vous informer sur ses fonctionnalités et sur les fichiers générés.
< Précédent
Suivant >
Terminer
Annuler

SAMPLE

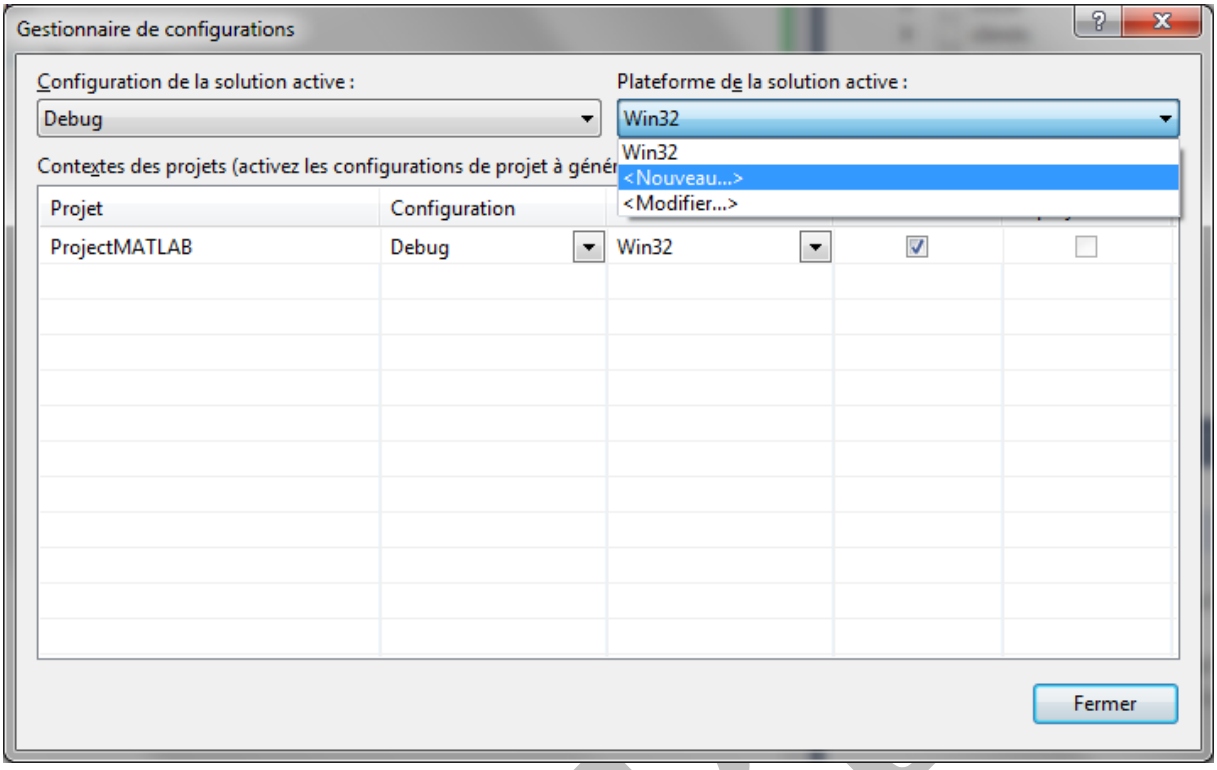

Assistant Application Win32 - ProjectMATLAB
Paramètres de l'application
Vue d'ensemble
Paramètres de l'application
Type d'application :
Application Windows
Application console
DLL
Bibliothèque statique
Options supplémentaires :
Projet vide
Exporter des symboles
En-tête précompilé
Vérifications SDL (Security Development Lifecycle)
Ajouter les fichiers d'en-tête courants pour :
ATL
MFC
< Précédent
Suivant >
Terminer
Annuler

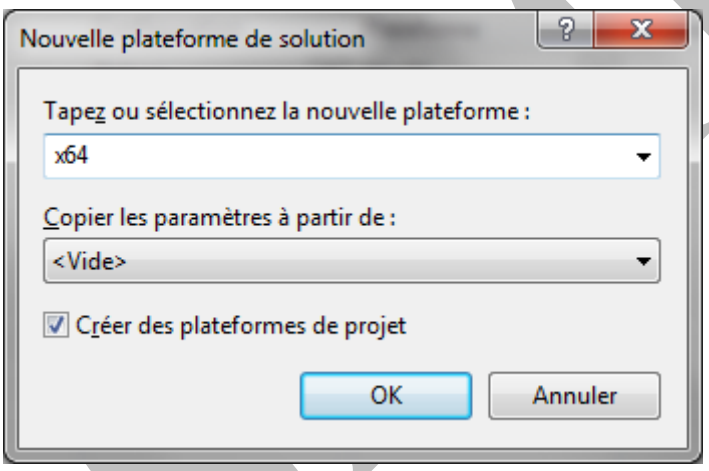

Windows Desktop
Lancement rapide (Ctrl+Q)
ÉBOGUER
ÉQUIPE
OUTILS
TEST
FENÊTRE
AIDE
main()
Attacher au processus... Ctrl+Alt+P
Connexion à la base de données...
Se connecter à Microsoft Azure...
SQL Server
Gestionnaire des extraits de code... Ctrl+K, Ctrl+B
Choisir des éléments de boîte à outils...
Gestionnaire de package NuGet
Extensions et mises à jour...
Créer un Guid
Errlook - Recherche d'erreurs
Outils externes...
Importation et exportation de paramètres...
Personnaliser...
Options...

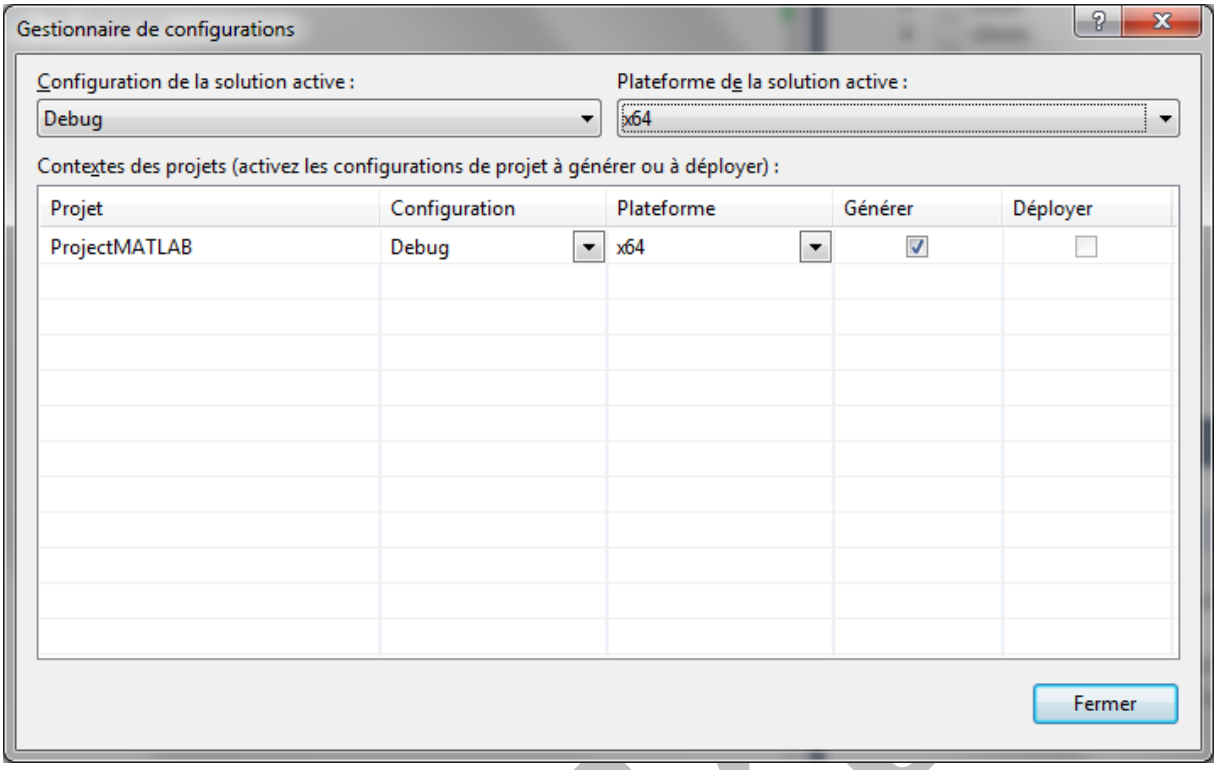

Options
Options de recherche (Ctrl+E)
Environnement
Projets et solutions
Contrôle de code source
Éditeur de texte
Débogage
Général
Fenêtre Sortie
Modifier & Continuer
Symboles
Concepteur Windows Forms
Création de modèles de texte
Gestionnaire de package NuGet
Outils de base de données
Outils de test de performances Web
Outils SQL Server
Performance Tools
Symboles
Emplacements du fichier de symboles (.pdb) :
Serveurs de symboles Microsoft
Mettre en cache les symboles dans ce répertoire :
C:\Users\VINCEN~1\AppData\Local\Temp\SymbolCache
Parcourir...
Charger tous les symboles
Vider le cache de symboles
Charger automatiquement les symboles pour :
Tous les modules, sauf exclus
Spécifier les modules exclus
Modules spécifiés uniquement
Spécifier les modules
OK
Annuler


s 2013 pour Windows Desktop
Lancement
NÉRER DÉBOGUER ÉQUIPE OUTILS TEST FENÊTRE AIDE
ébogueur Windows local
Debug
Win32
Win32
Gestionnaire de configurations...
obale)
Rechercher dans l'Exp

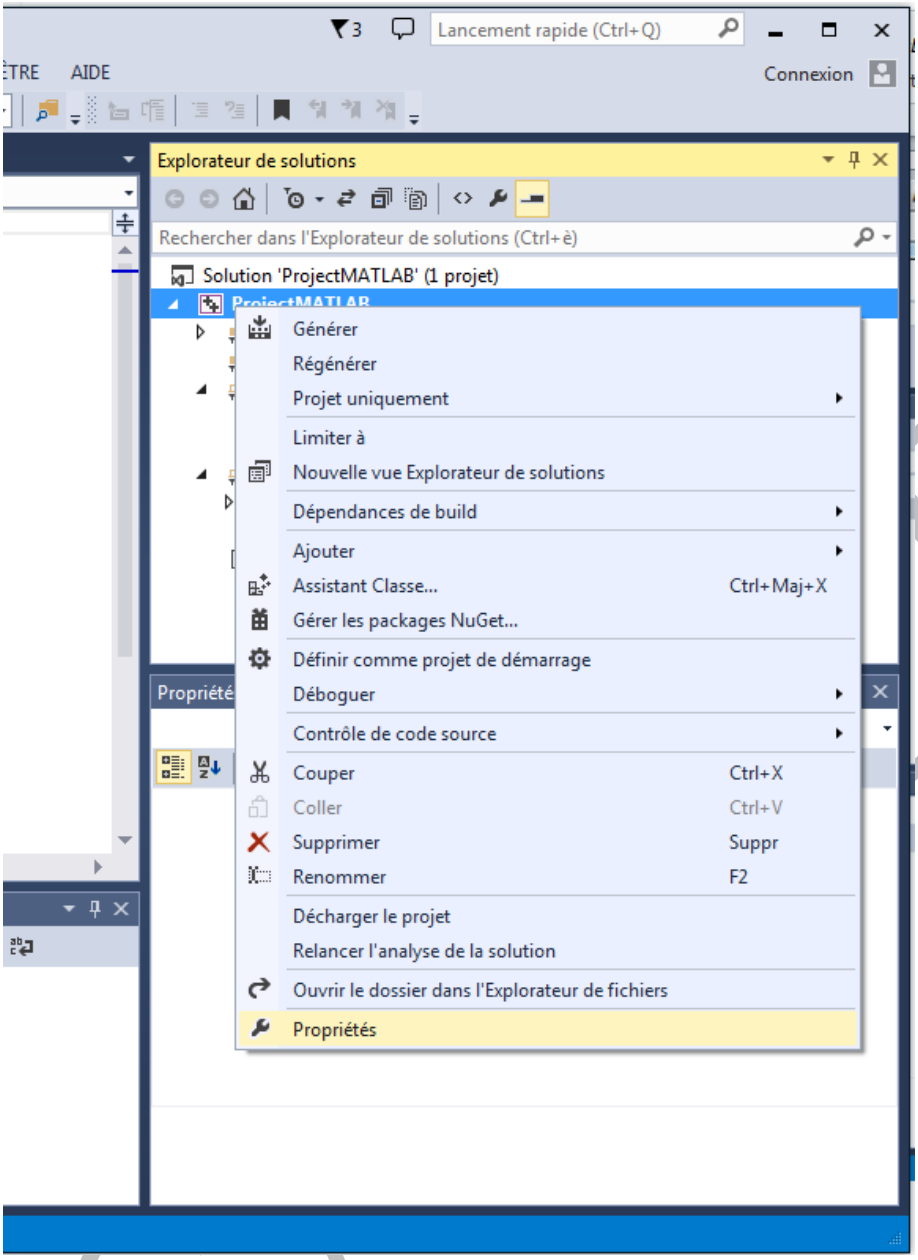
SAMPLE

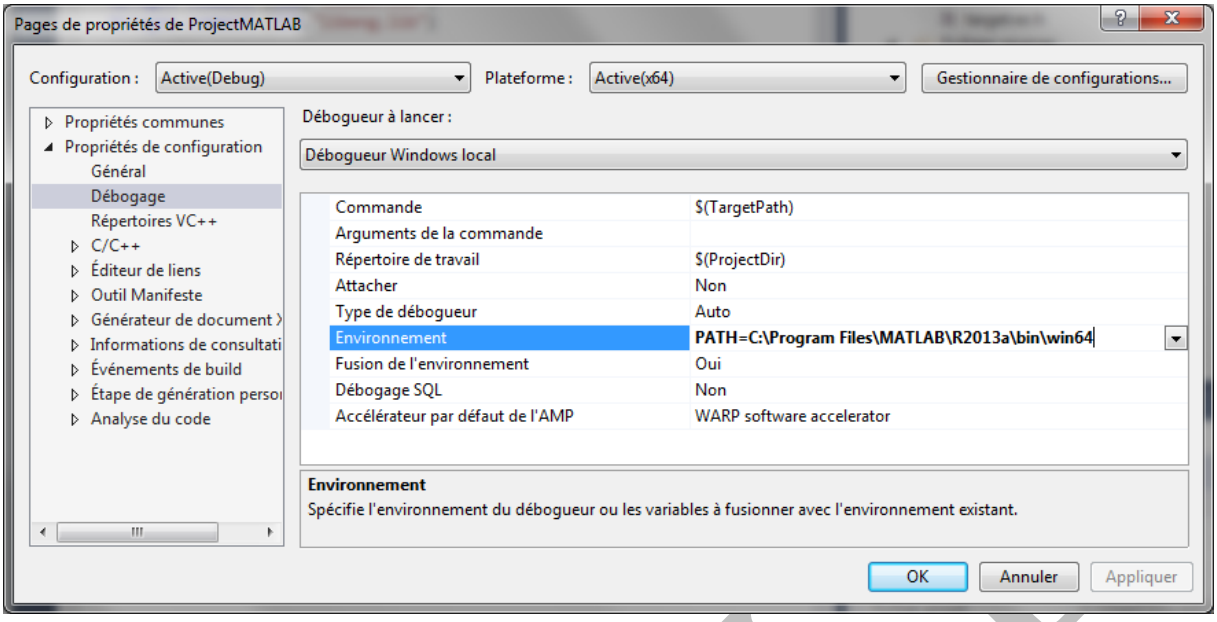

Gestionnaire de configurations
Configuration de la solution active :
Debug
Plateforme de la solution active :
Win32
Win32
<Nouveau...>
<Modifier...>
Contextes des projets (activez les configurations de projet à génér
Projet
Configuration
ProjectMATLAB
Debug
Win32
Fermer

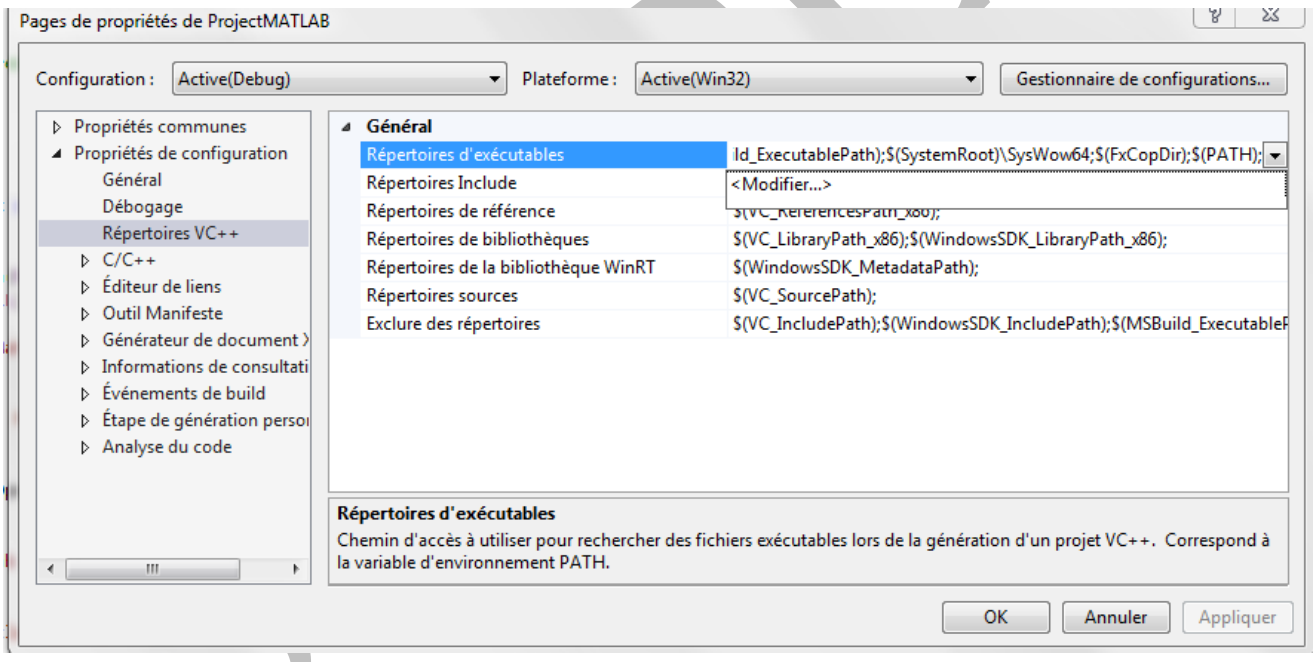

Nouvelle plateforme de solution
Tapez ou sélectionnez la nouvelle plateforme :
x64
Copier les paramètres à partir de :
<Vide>
Créer des plateformes de projet
OK
Annuler

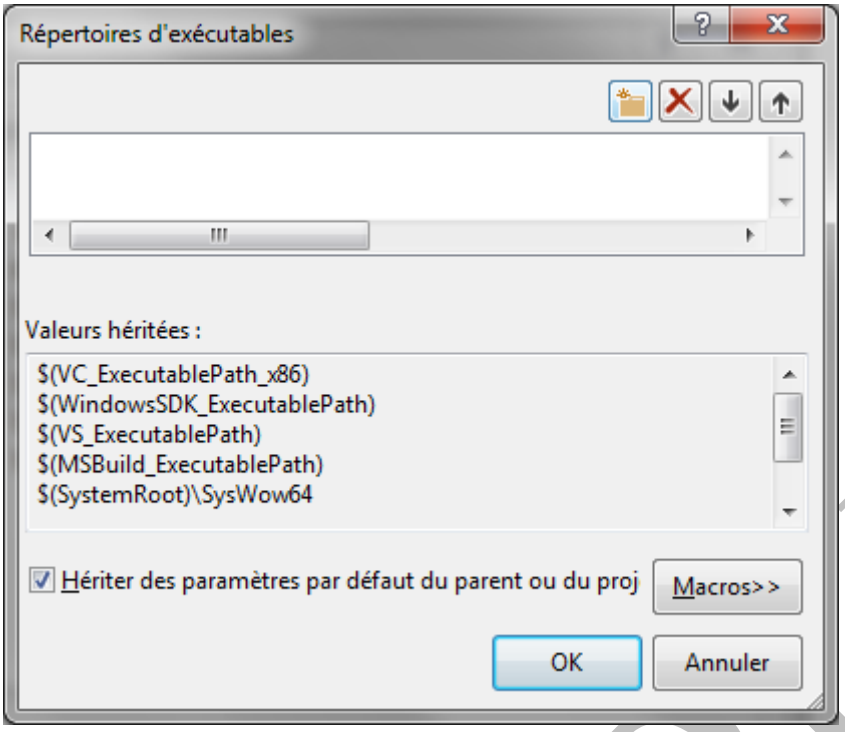

Gestionnaire de configurations
Configuration de la solution active :
Debug
Plateforme de la solution active :
x64
Contextes des projets (activez les configurations de projet à générer ou à déployer) :
Projet
Configuration
Plateforme
Générer
Déployer
ProjectMATLAB
Debug
x64
Fermer

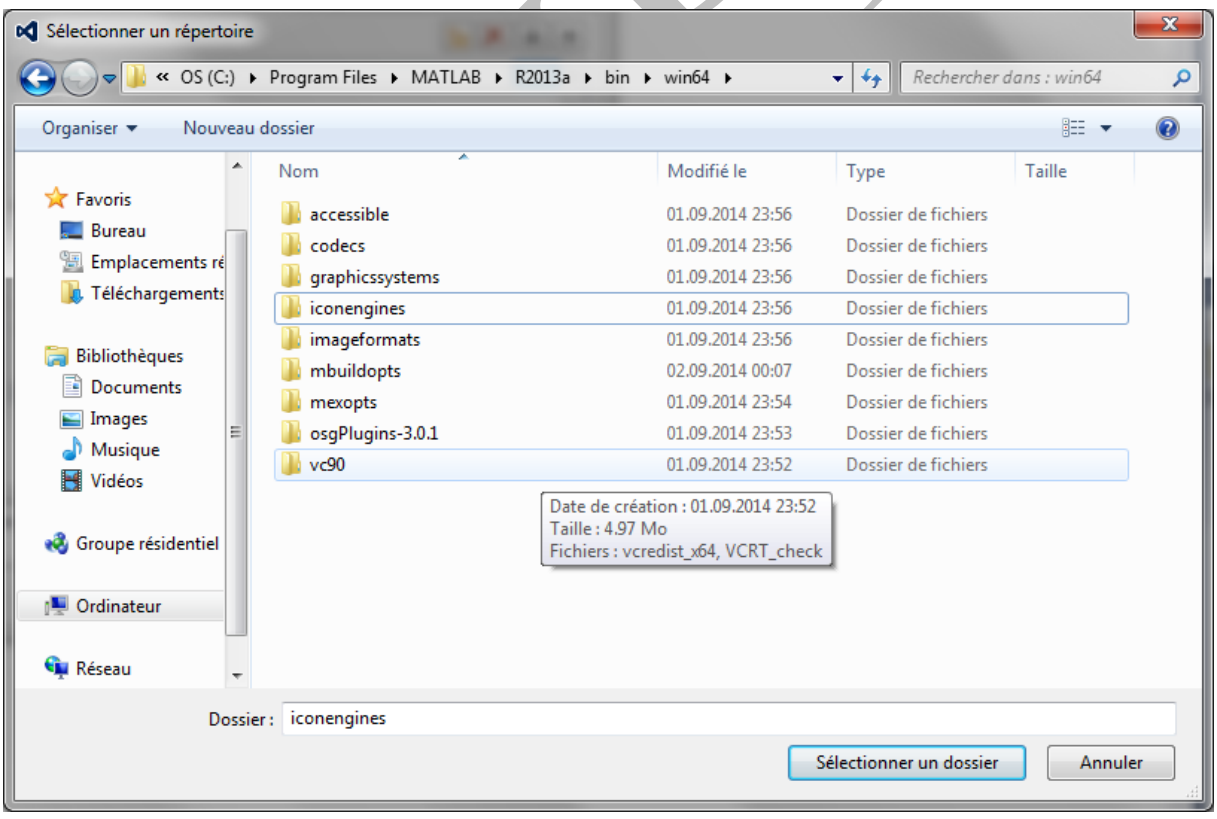
SAMPLE

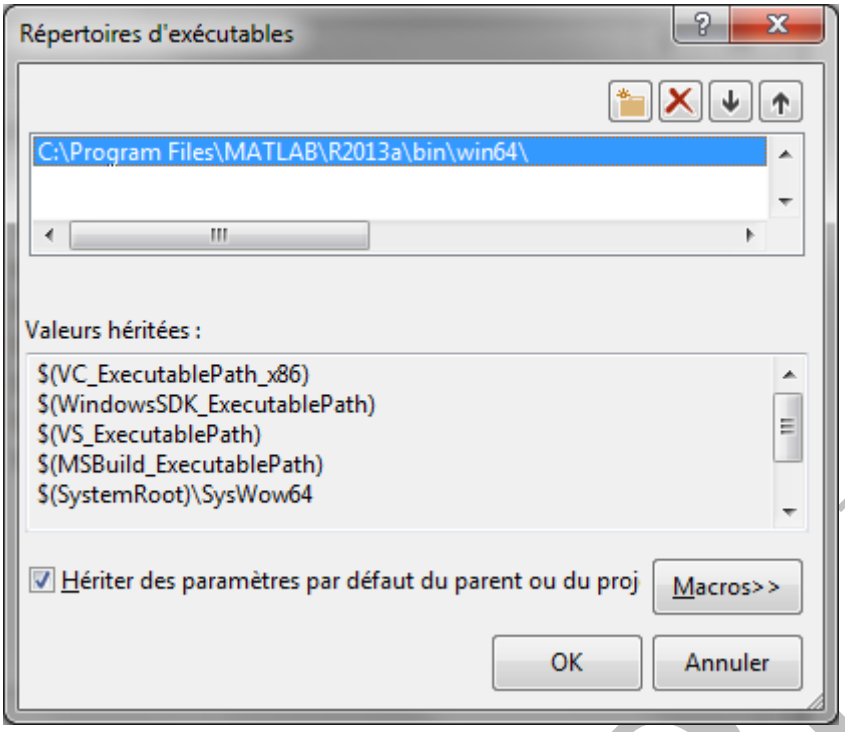

Lancement rapide (Ctrl+Q)
AIDE
Connexion
Explorateur de solutions
Rechercher dans l'Explorateur de solutions (Ctrl+è)
Solution 'ProjectMATLAB' (1 projet)
Générer
Régénérer
Projet uniquement
Limiter à
Nouvelle vue Explorateur de solutions
Dépendances de build
Ajouter
Assistant Classe... Ctrl+Maj+X
Gérer les packages NuGet...
Définir comme projet de démarrage
Déboguer
Contrôle de code source
Couper Ctrl+X
Coller Ctrl+V
Supprimer Suppr
Renommer F2
Décharger le projet
Relancer l'analyse de la solution
Ouvrir le dossier dans l'Explorateur de fichiers
Propriétés

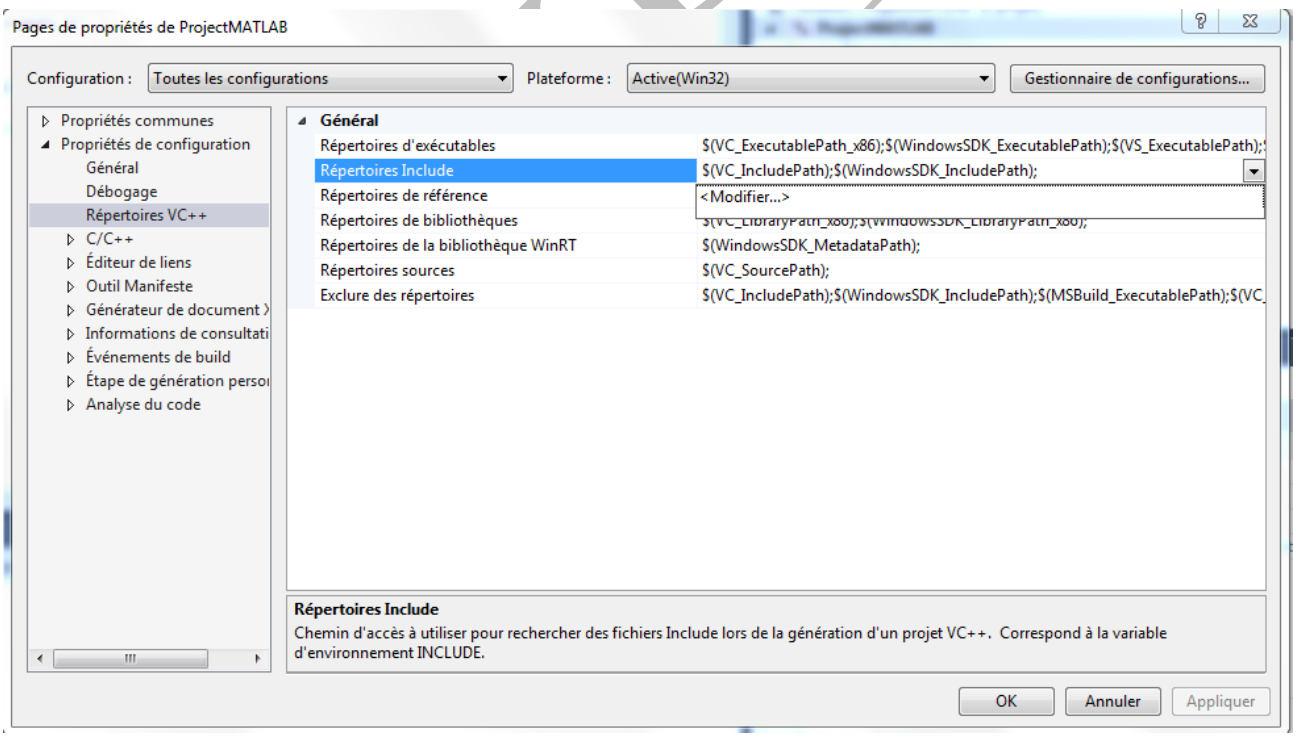

Pages de propriétés de ProjectMATLAB
Configuration : Active(Debug)
Plateforme : Active(x64)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Outil Manifeste
Générateur de document
Informations de consultati
Événements de build
Étape de génération perso
Analyse du code
Débogueur à lancer :
Débogueur Windows local
Commande $(TargetPath)
Arguments de la commande
Répertoire de travail $(ProjectDir)
Attacher Non
Type de débogueur Auto
Environnement PATH=C:\Program Files\MATLAB\R2013a\bin\win64
Fusion de l'environnement Oui
Débogage SQL Non
Accélérateur par défaut de l'AMP WARP software accelerator
Environnement
Spécifie l'environnement du débogueur ou les variables à fusionner avec l'environnement existant.
OK
Annuler
Appliquer

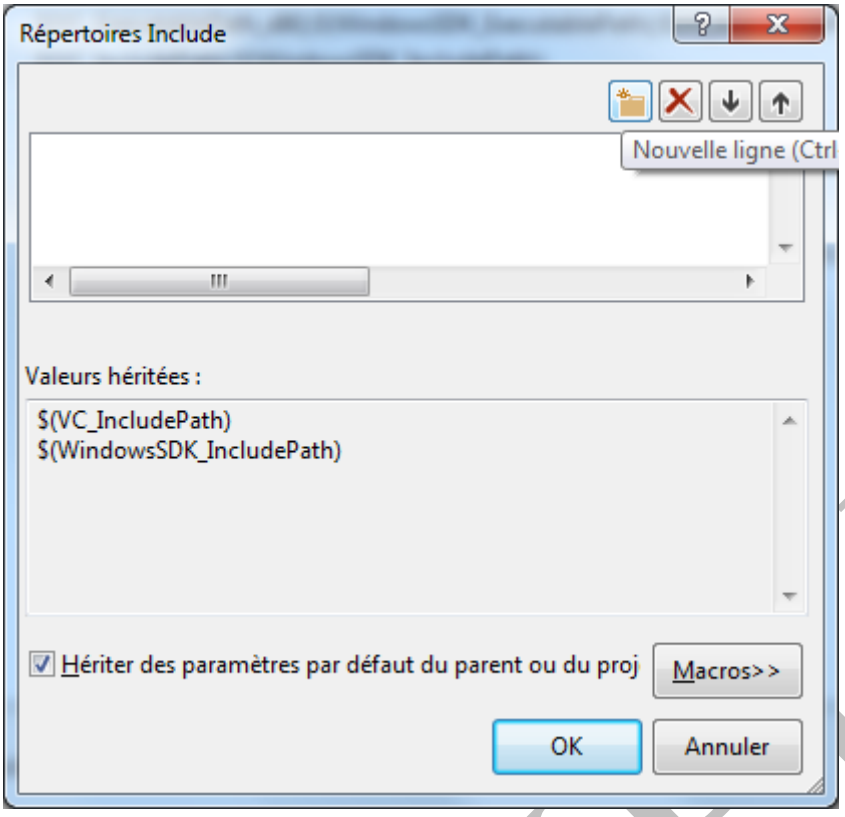

Pages de propriétés de ProjectMATLAB
Configuration : Active(Debug)
Plateforme : Active(Win32)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Outil Manifeste
Générateur de document
Informations de consultati
Événements de build
Étape de génération perso
Analyse du code
Général
Répertoires d'exécutables Id_ExecutablePath);$(SystemRoot)\SysWow64;$(FxCopDir);$(PATH);
<Modifier...>
Répertoires Include
Répertoires de référence
Répertoires de bibliothèques $(VC_LibraryPath_x86);$(WindowsSDK_LibraryPath_x86);
Répertoires de la bibliothèque WinRT $(WindowsSDK_MetadataPath);
Répertoires sources $(VC_SourcePath);
Exclure des répertoires $(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutableP
Répertoires d'exécutables
Chemin d'accès à utiliser pour rechercher des fichiers exécutables lors de la génération d'un projet VC++. Correspond à la variable d'environnement PATH.
OK
Annuler
Appliquer

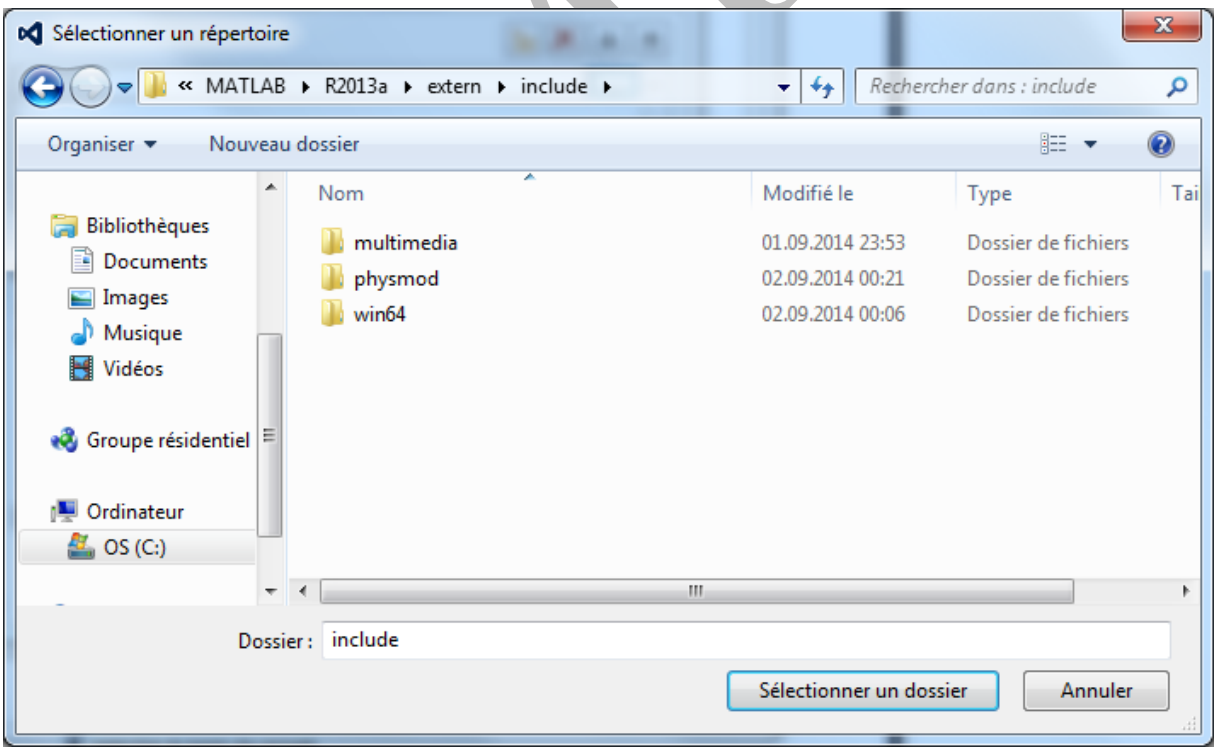

Répertoires d'exécutables
Valeurs héritées :
$(VC_ExecutablePath_x86)
$(WindowsSDK_ExecutablePath)
$(VS_ExecutablePath)
$(MSBuild_ExecutablePath)
$(SystemRoot)\SysWow64
Hériter des paramètres par défaut du parent ou du proj
Macros>>
OK
Annuler

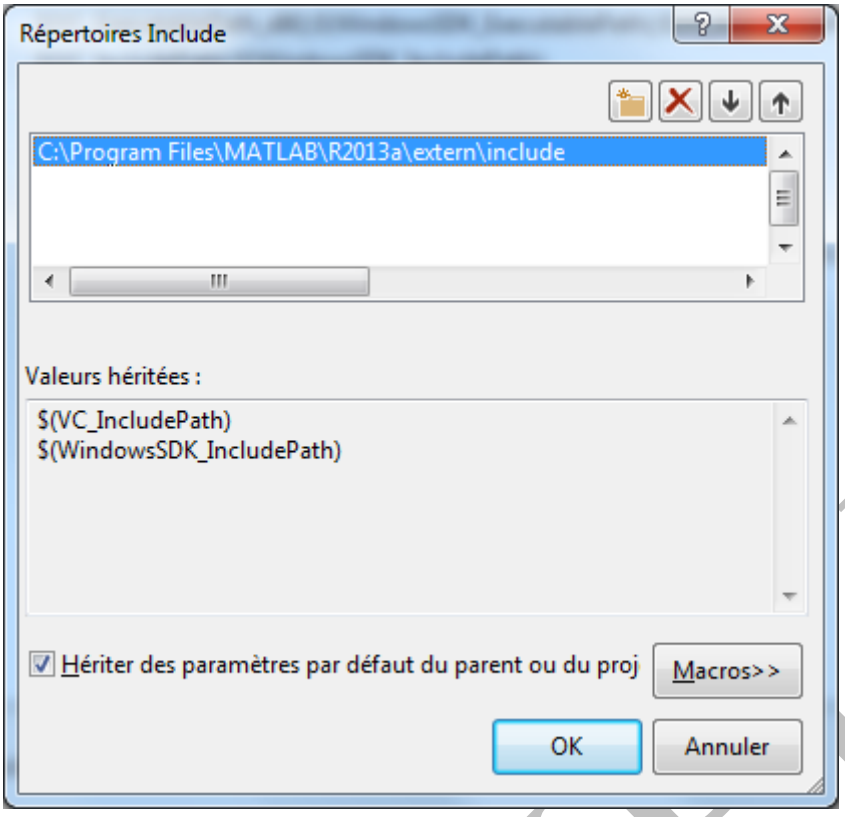

Sélectionner un répertoire
OS (C:) ▸ Program Files ▸ MATLAB ▸ R2013a ▸ bin ▸ win64 ▸
Rechercher dans : win64
Organiser
Nouveau dossier
Favoris
Bureau
Emplacements ré
Téléchargements
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
Réseau
Nom
Modifié le
Type
Taille
accessible 01.09.2014 23:56 Dossier de fichiers
codecs 01.09.2014 23:56 Dossier de fichiers
graphicssystems 01.09.2014 23:56 Dossier de fichiers
iconengines 01.09.2014 23:56 Dossier de fichiers
imageformats 01.09.2014 23:56 Dossier de fichiers
mbuildopts 02.09.2014 00:07 Dossier de fichiers
mexopts 01.09.2014 23:54 Dossier de fichiers
osgPlugins-3.0.1 01.09.2014 23:53 Dossier de fichiers
vc90 01.09.2014 23:52 Dossier de fichiers
Date de création : 01.09.2014 23:52
Taille : 4.97 Mo
Fichiers : vcredist_x64, VCRT_check
Dossier : iconengines
Sélectionner un dossier
Annuler


Répertoires d'exécutables
C:\Program Files\MATLAB\R2013a\bin\win64\
Valeurs héritées :
$(VC_ExecutablePath_x86)
$(WindowsSDK_ExecutablePath)
$(VS_ExecutablePath)
$(MSBuild_ExecutablePath)
$(SystemRoot)\SysWow64
Hériter des paramètres par défaut du parent ou du proj
Macros>>
OK
Annuler


Pages de propriétés de ProjectMATLAB
Configuration : Toutes les configurations
Plateforme : Active(Win32)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Outil Manifeste
Générateur de document
Informations de consultati
Événements de build
Étape de génération perso
Analyse du code
Général
Répertoires d'exécutables
$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);
Répertoires Include
$(VC_IncludePath);$(WindowsSDK_IncludePath);
Répertoires de référence
<Modifier...>
Répertoires de bibliothèques
Répertoires de la bibliothèque WinRT
$(WindowsSDK_MetadataPath);
Répertoires sources
$(VC_SourcePath);
Exclure des répertoires
$(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC
Répertoires Include
Chemin d'accès à utiliser pour rechercher des fichiers Include lors de la génération d'un projet VC++. Correspond à la variable d'environnement INCLUDE.
OK
Annuler
Appliquer

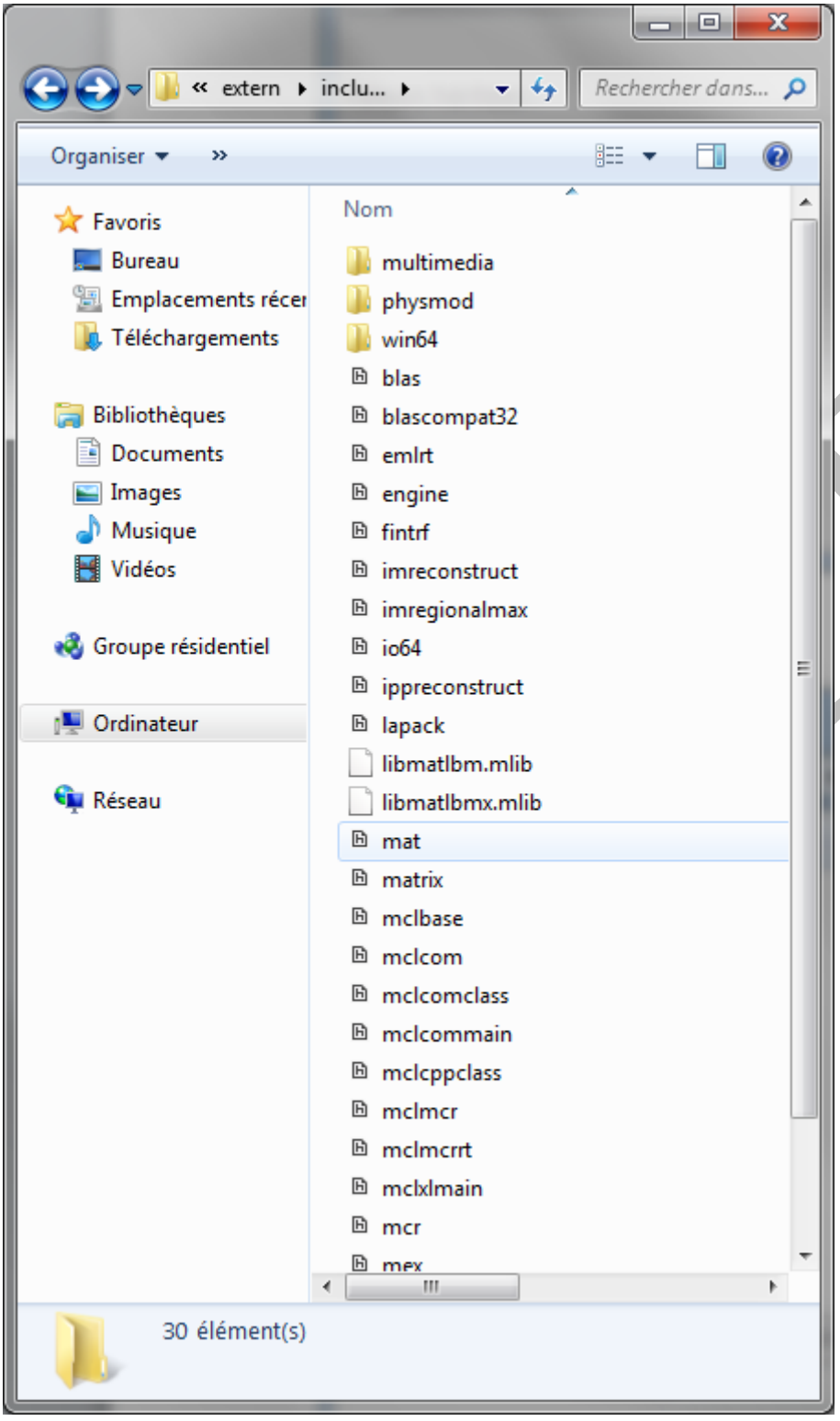

Répertoires Include
Nouvelle ligne (Ctrl
Valeurs héritées :
$(VC_IncludePath)
$(WindowsSDK_IncludePath)
Hériter des paramètres par défaut du parent ou du proj
Macros>>
OK
Annuler

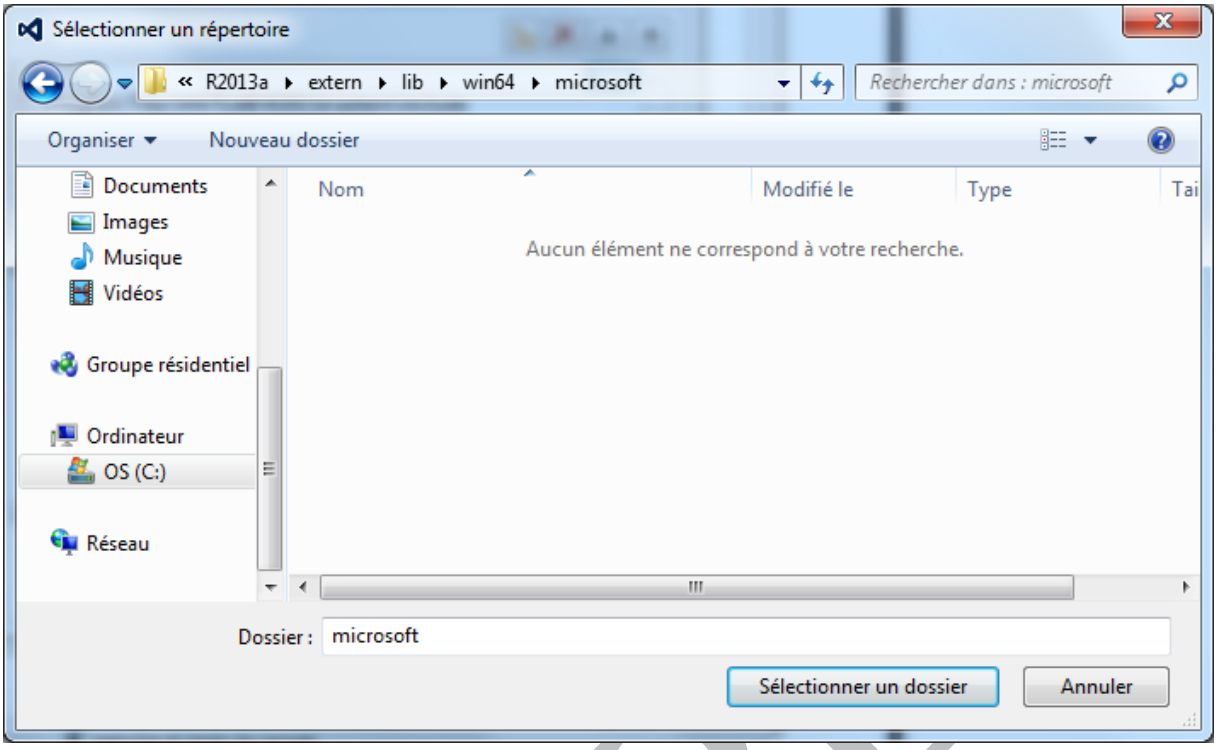

Sélectionner un répertoire
« MATLAB ▸ R2013a ▸ extern ▸ include ▸
Rechercher dans : include
Organiser
Nouveau dossier
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
OS (C:)
Nom
Modifié le
Type
multimedia 01.09.2014 23:53 Dossier de fichiers
physmod 02.09.2014 00:21 Dossier de fichiers
win64 02.09.2014 00:06 Dossier de fichiers
Dossier : include
Sélectionner un dossier
Annuler

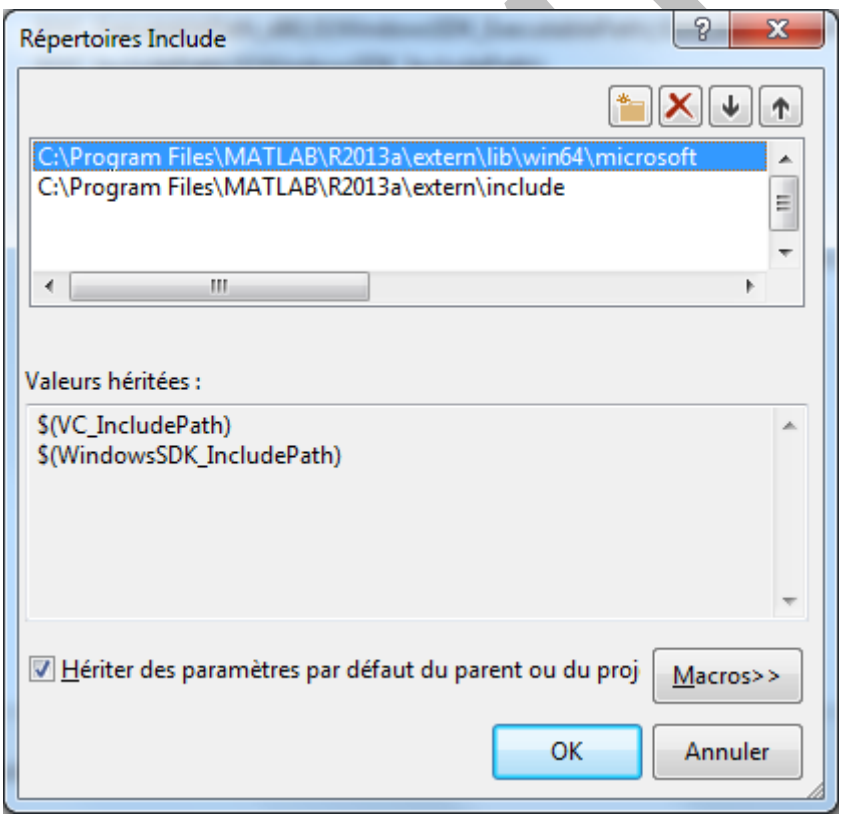


For info that's the content :

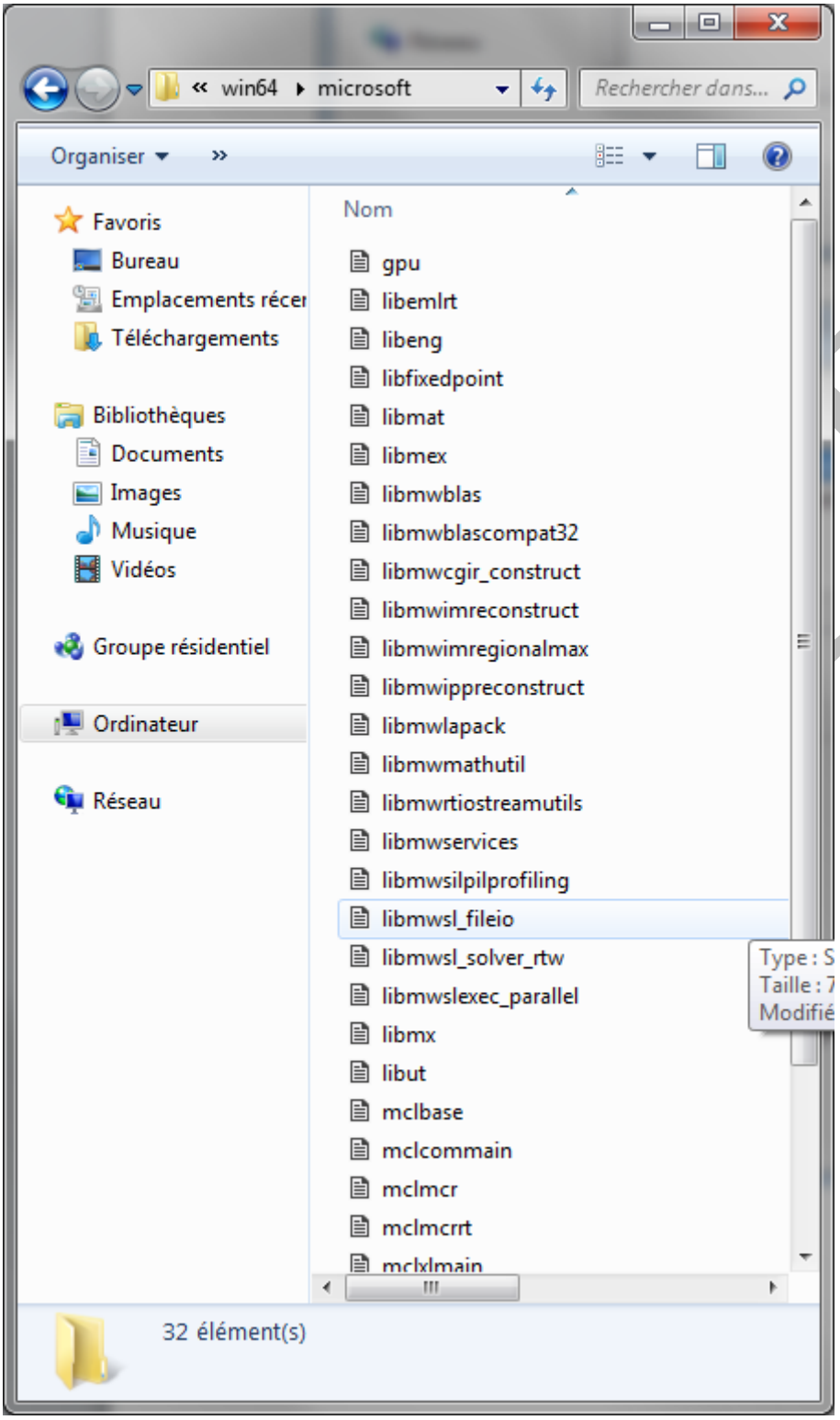
SAMPLE

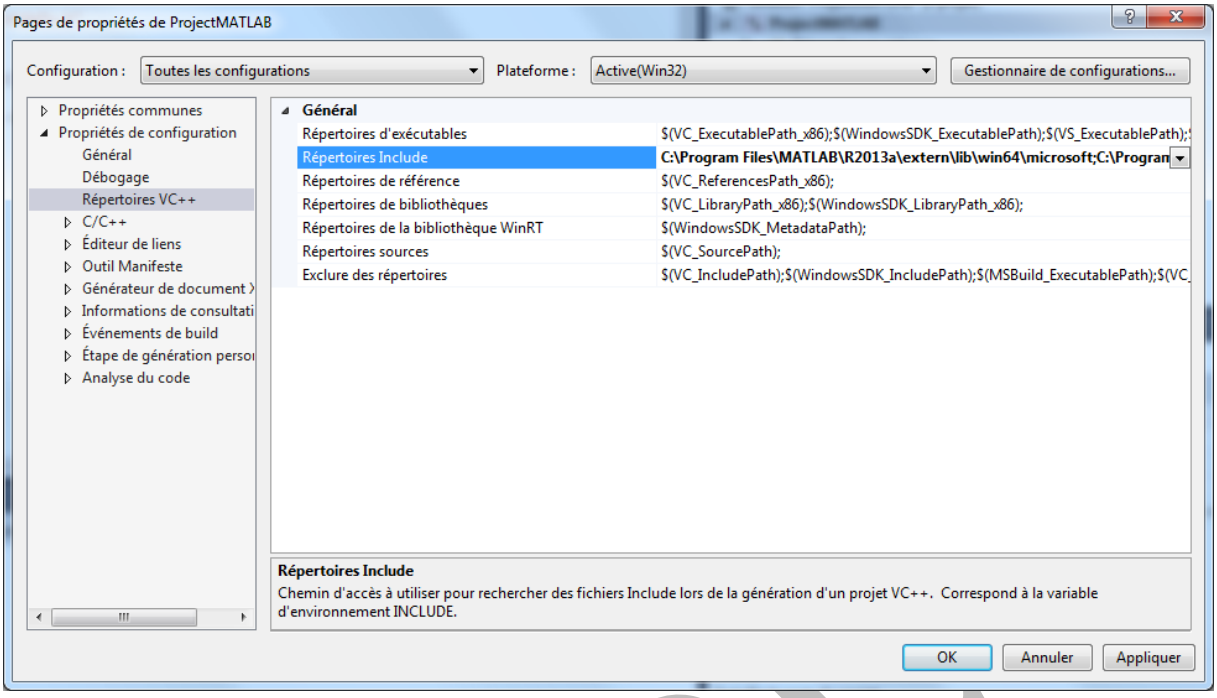

« extern ▸ inclu... ▸
Rechercher dans...
Organiser
Favoris
Bureau
Emplacements récer
Téléchargements
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
Réseau
Nom
multimedia
physmod
win64
blas
blascompat32
emlrt
engine
fintrf
imreconstruct
imregionalmax
io64
ippreconstruct
lapack
libmatlbm.mlib
libmatlbmx.mlib
mat
matrix
mclbase
mclcom
mclcomclass
mclcommain
mclcppclass
mclmcr
mclmcrrt
mclxlmain
mcr
mex
30 élément(s)

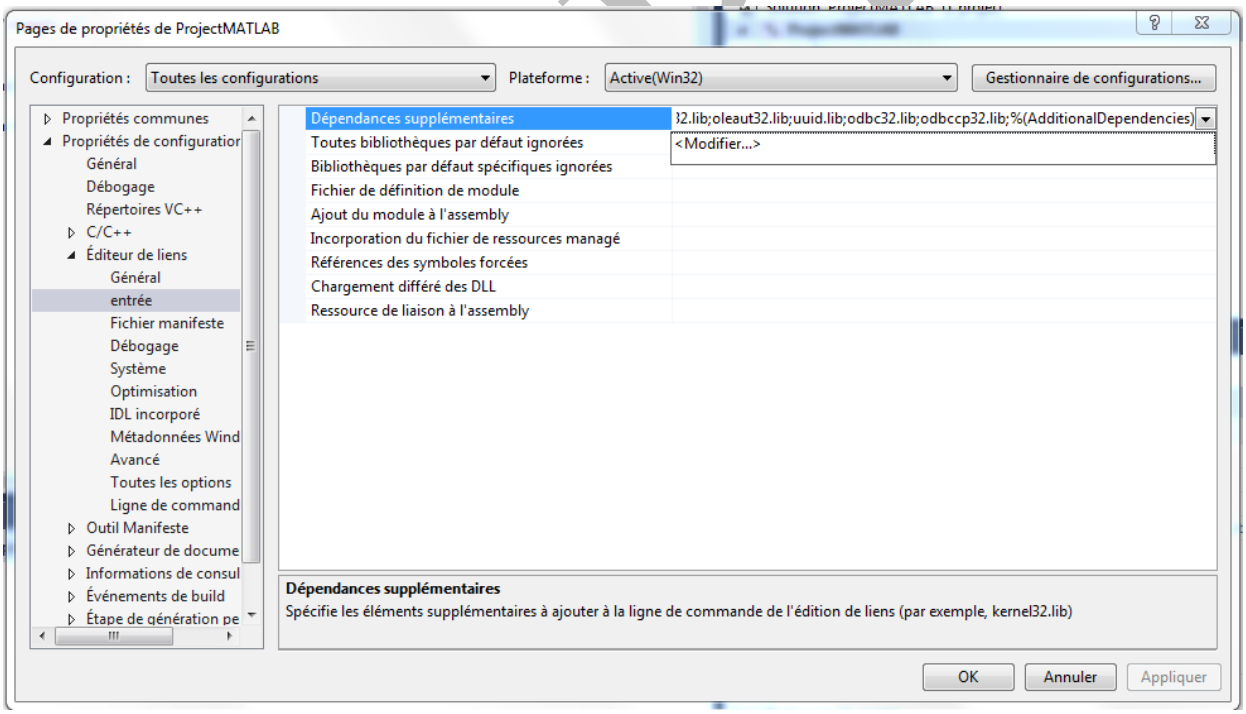


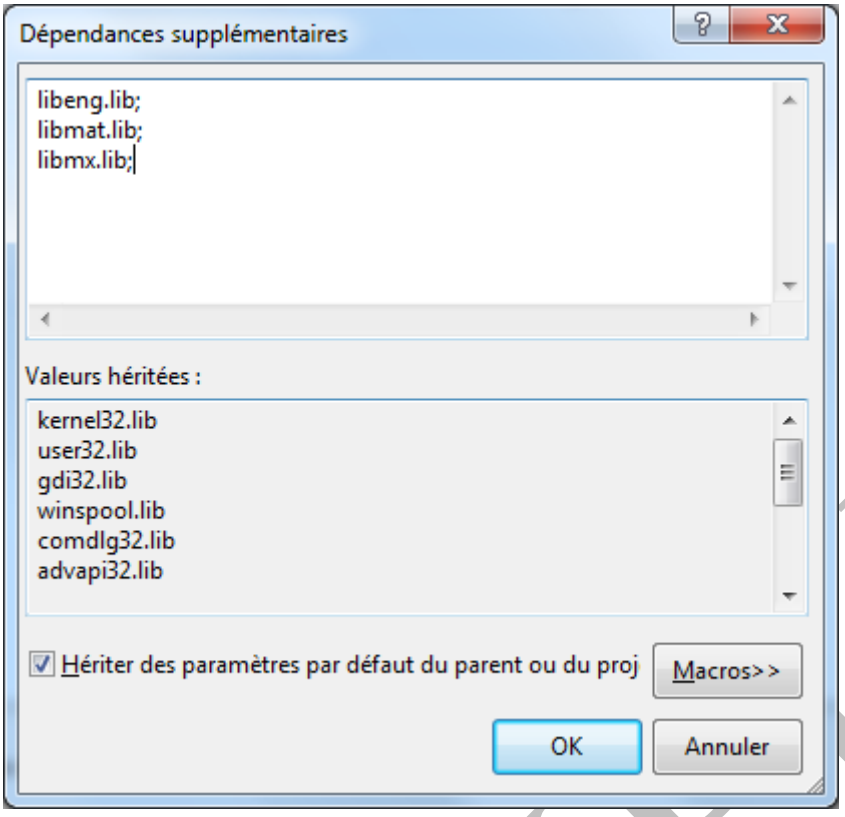


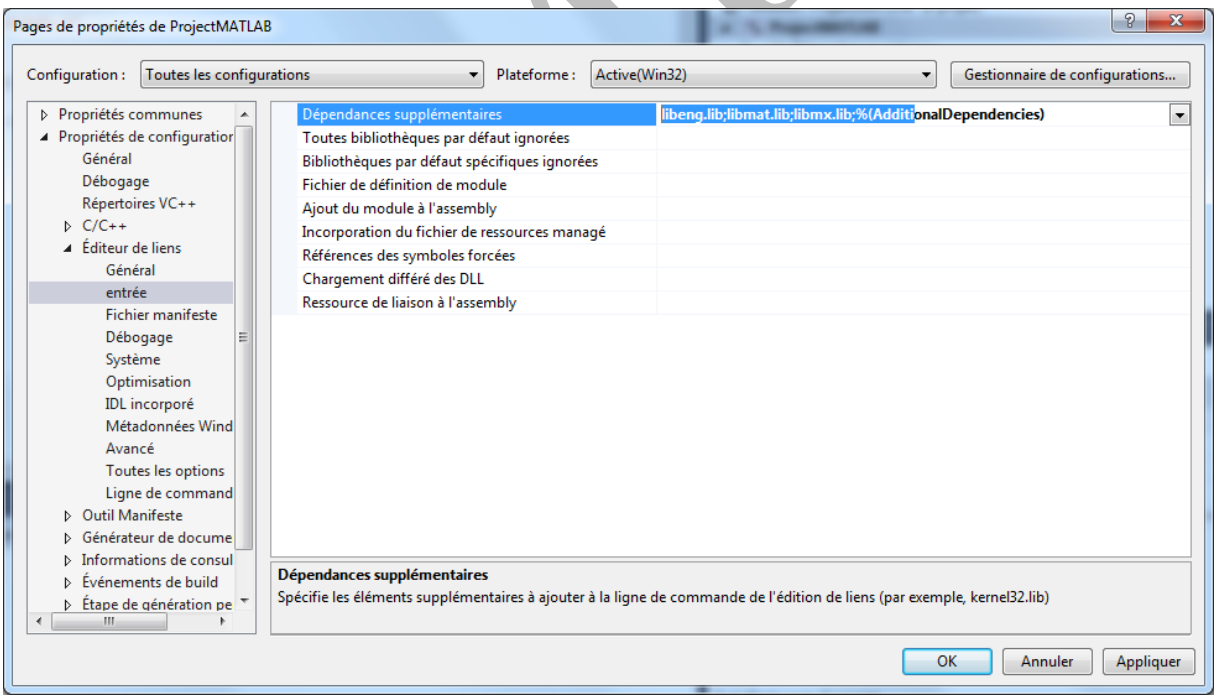
For info that's the content :

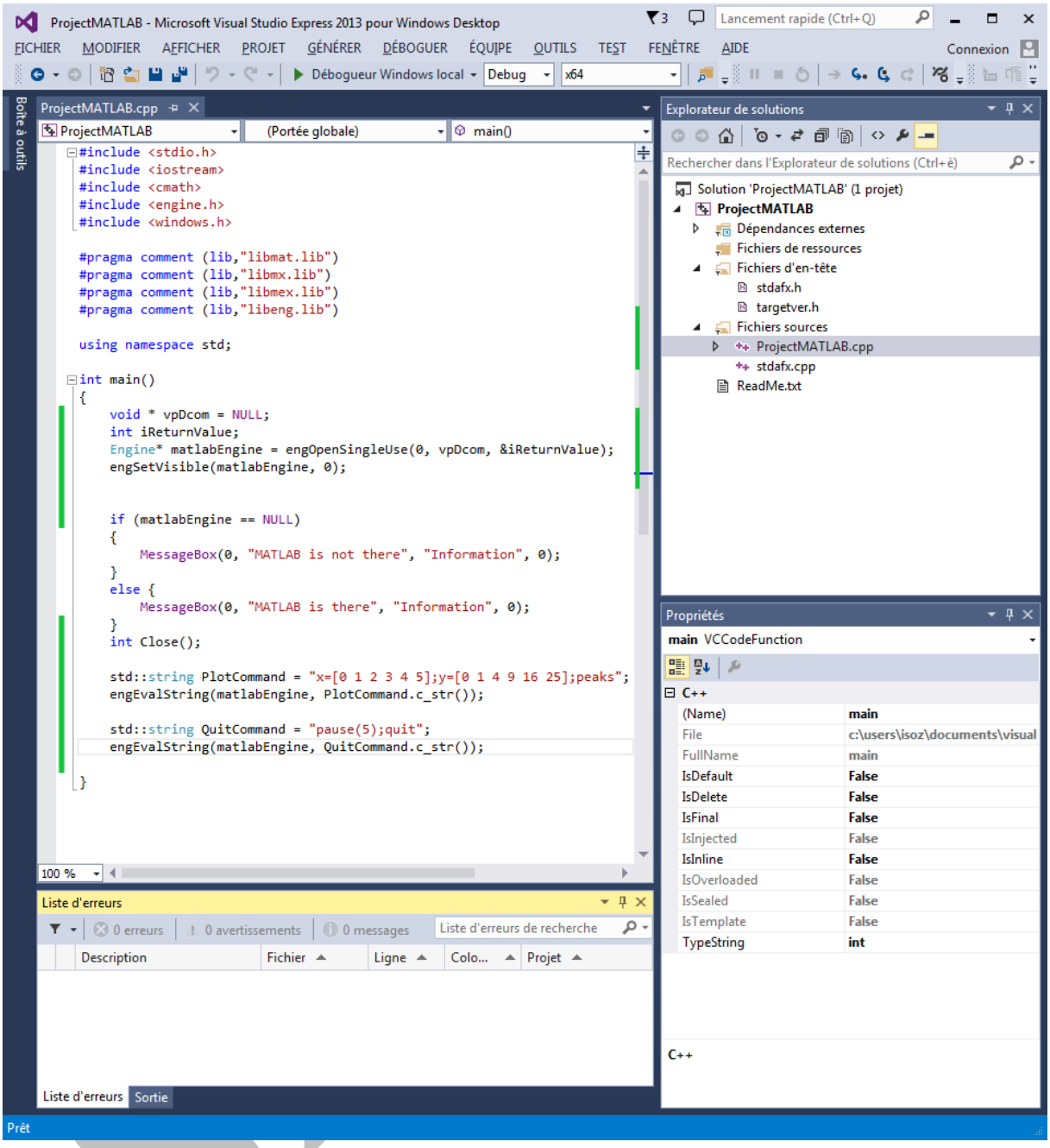

win64 ▸ microsoft
Rechercher dans...
Organiser
Favoris
Bureau
Emplacements récer
Téléchargements
Bibliothèques
Documents
Images
Musique
Vidéos
Groupe résidentiel
Ordinateur
Réseau
Nom
gpu
libemlrt
libeng
libfixedpoint
libmat
libmex
libmwblas
libmwblascompat32
libmwcgir_construct
libmwimreconstruct
libmwimregionalmax
libmwippreconstruct
libmwlapack
libmwmathutil
libmwrtiostreamutils
libmwservices
libmwsilpilprofiling
libmwsl_fileio
libmwsl_solver_rtw
libmwslexec_parallel
libmx
libut
mclbase
mclcommain
mclmcr
mclmcrrt
32 élément(s)

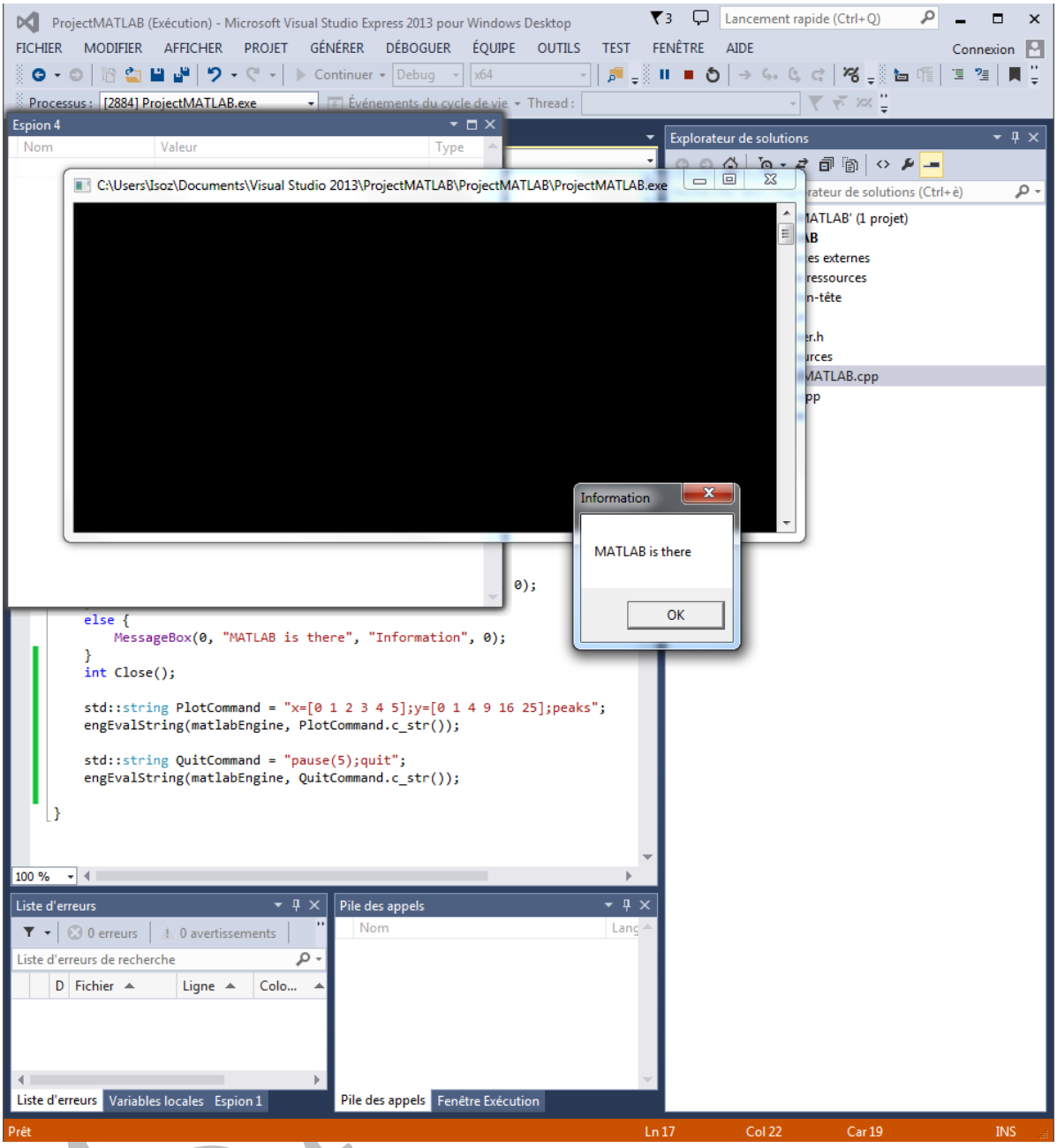

Pages de propriétés de ProjectMATLAB
Configuration : Toutes les configurations
Plateforme : Active(Win32)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Outil Manifeste
Générateur de document
Informations de consultati
Événements de build
Étape de génération perso
Analyse du code
Général
Répertoires d'exécutables
$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);
Répertoires Include
C:\Program Files\MATLAB\R2013a\extern\lib\win64\microsoft;C:\Program
Répertoires de référence
$(VC_ReferencesPath_x86);
Répertoires de bibliothèques
$(VC_LibraryPath_x86);$(WindowsSDK_LibraryPath_x86);
Répertoires de la bibliothèque WinRT
$(WindowsSDK_MetadataPath);
Répertoires sources
$(VC_SourcePath);
Exclure des répertoires
$(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC_
Répertoires Include
Chemin d'accès à utiliser pour rechercher des fichiers Include lors de la génération d'un projet VC++. Correspond à la variable d'environnement INCLUDE.
OK
Annuler
Appliquer

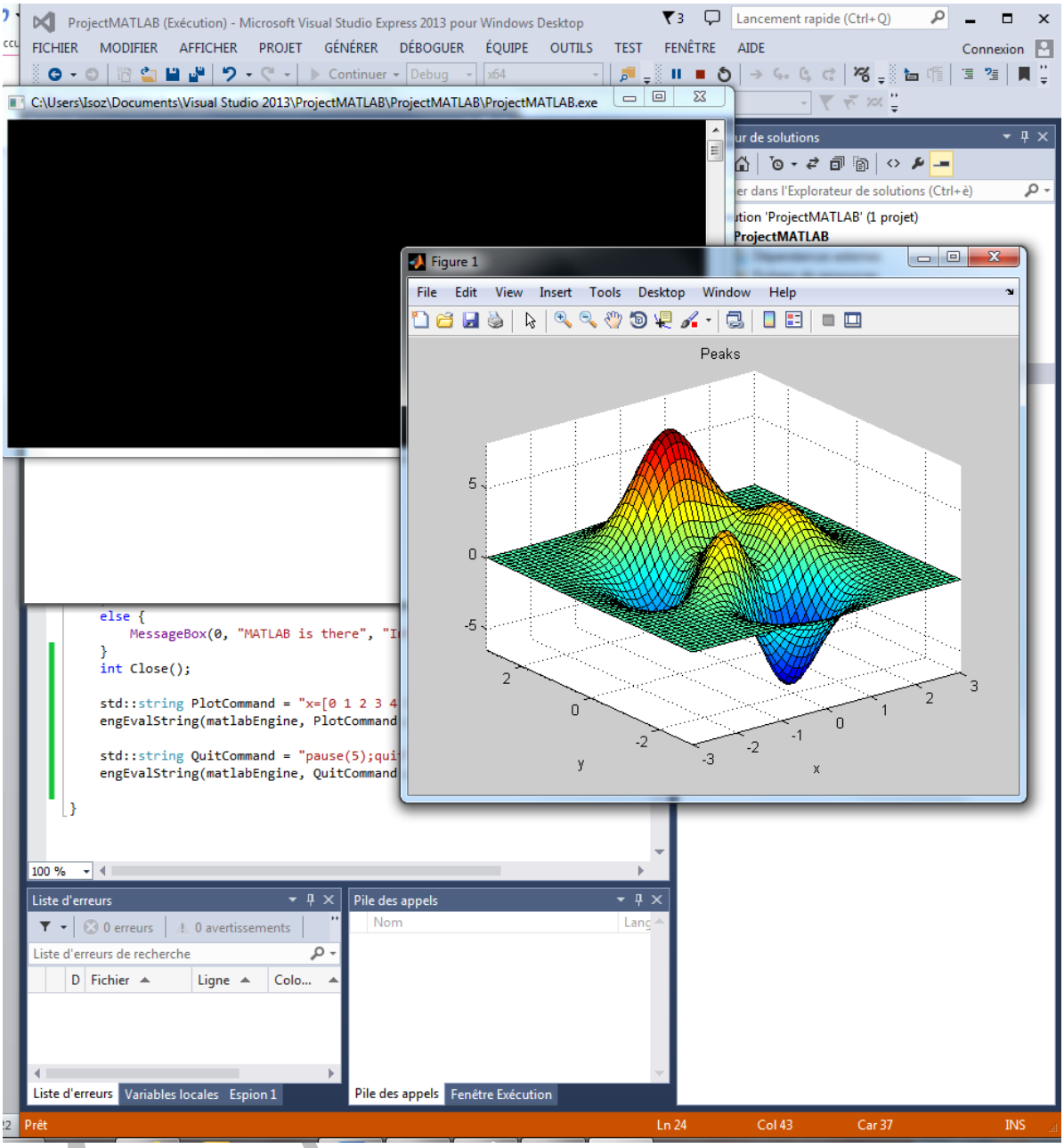

Pages de propriétés de ProjectMATLAB
Configuration : Toutes les configurations
Plateforme : Active(Win32)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Général
entrée
Fichier manifeste
Débogage
Système
Optimisation
IDL incorporé
Métadonnées Wind
Avancé
Toutes les options
Ligne de command
Outil Manifeste
Générateur de docume
Informations de consul
Événements de build
Étape de génération pe
Dépendances supplémentaires
32.lib;oleaut32.lib;uuid.lib;odbc32.lib;odbccp32.lib;%(AdditionalDependencies)
Toutes bibliothèques par défaut ignorées
<Modifier...>
Bibliothèques par défaut spécifiques ignorées
Fichier de définition de module
Ajout du module à l'assembly
Incorporation du fichier de ressources managé
Références des symboles forcées
Chargement différé des DLL
Ressource de liaison à l'assembly
Dépendances supplémentaires
Spécifie les éléments supplémentaires à ajouter à la ligne de commande de l'édition de liens (par exemple, kernel32.lib)
OK
Annuler
Appliquer

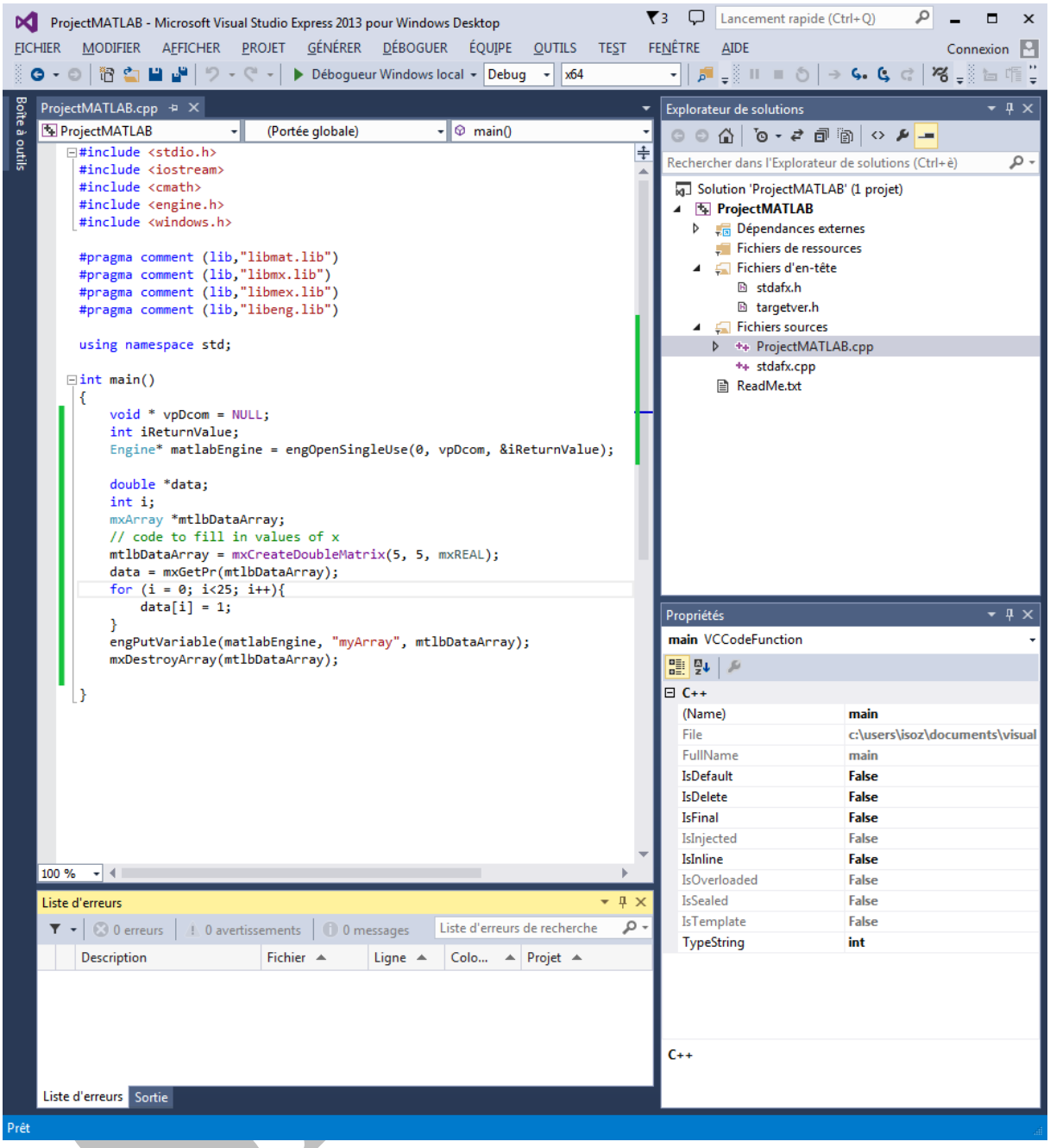

Dépendances supplémentaires
libeng.lib;
libmat.lib;
libmx.lib;
Valeurs héritées :
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
advapi32.lib
Hériter des paramètres par défaut du parent ou du proj
Macros>>
OK
Annuler

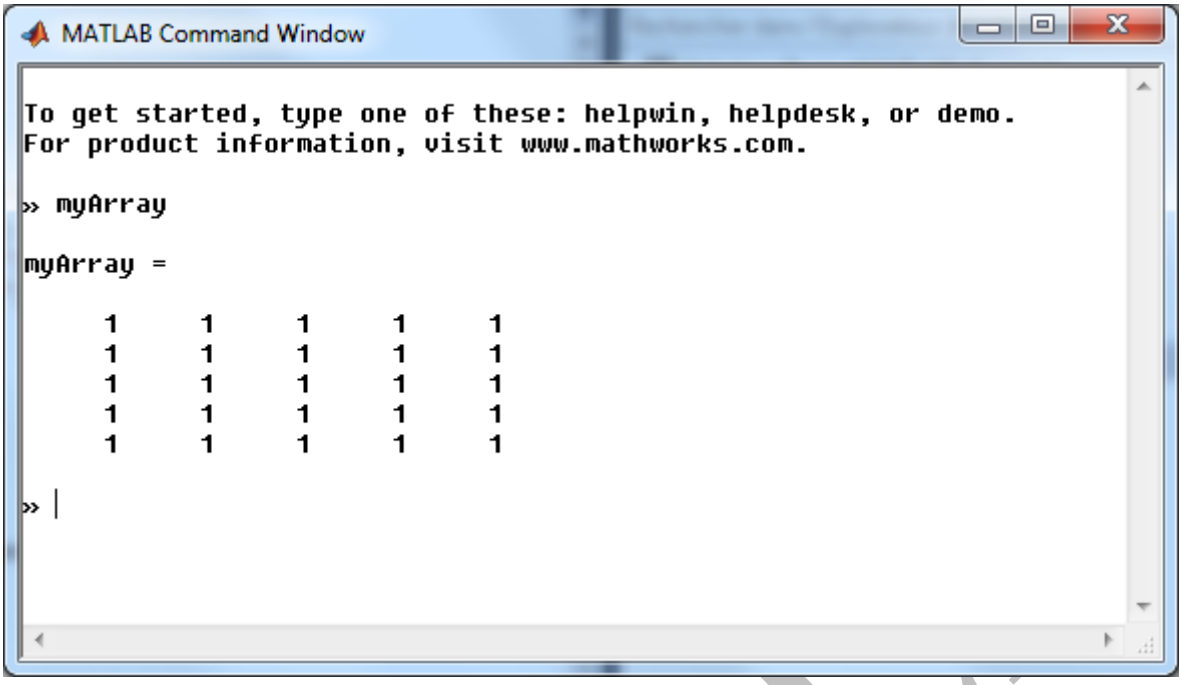

Pages de propriétés de ProjectMATLAB
Configuration :
Toutes les configurations
Plateforme :
Active(Win32)
Gestionnaire de configurations...
Propriétés communes
Propriétés de configuration
Général
Débogage
Répertoires VC++
C/C++
Éditeur de liens
Général
entrée
Fichier manifeste
Débogage
Système
Optimisation
IDL incorporé
Métadonnées Wind
Avancé
Toutes les options
Ligne de command
Outil Manifeste
Générateur de docume
Informations de consul
Événements de build
Étape de génération pe
Dépendances supplémentaires
libeng.lib;libmat.lib;libmx.lib;%(AdditionalDependencies)
Toutes bibliothèques par défaut ignorées
Bibliothèques par défaut spécifiques ignorées
Fichier de définition de module
Ajout du module à l'assembly
Incorporation du fichier de ressources managé
Références des symboles forcées
Chargement différé des DLL
Ressource de liaison à l'assembly
Dépendances supplémentaires
Spécifie les éléments supplémentaires à ajouter à la ligne de commande de l'édition de liens (par exemple, kernel32.lib)
OK
Annuler
Appliquer


ProjectMATLAB - Microsoft Visual Studio Express 2013 pour Windows Desktop
FICHIER MODIFIER AFFICHER PROJET GÉNÉRER DÉBOGUER ÉQUIPE OUTILS TEST FENÊTRE AIDE
Connexion
Débogueur Windows local
Debug
x64
Boîte à outils
ProjectMATLAB.cpp
ProjectMATLAB
(Portée globale)
main()
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <engine.h>
#include <windows.h>
#pragma comment (lib,"libmat.lib")
#pragma comment (lib,"libmx.lib")
#pragma comment (lib,"libmex.lib")
#pragma comment (lib,"libeng.lib")
using namespace std;
int main()
{
void * vpDcom = NULL;
int iReturnValue;
Engine* matlabEngine = engOpenSingleUse(0, vpDcom, &iReturnValue);
engSetVisible(matlabEngine, 0);
if (matlabEngine == NULL)
{
MessageBox(0, "MATLAB is not there", "Information", 0);
}
else {
MessageBox(0, "MATLAB is there", "Information", 0);
}
int Close();
std::string PlotCommand = "x=[0 1 2 3 4 5];y=[0 1 4 9 16 25];peaks";
engEvalString(matlabEngine, PlotCommand.c_str());
std::string QuitCommand = "pause(5);quit";
engEvalString(matlabEngine, QuitCommand.c_str());
}
100 %
Liste d'erreurs
0 erreurs
0 avertissements
0 messages
Liste d'erreurs de recherche
Description
Fichier
Ligne
Colo...
Projet
Liste d'erreurs
Sortie
Explorateur de solutions
Rechercher dans l'Explorateur de solutions (Ctrl+è)
Solution 'ProjectMATLAB' (1 projet)
ProjectMATLAB
Dépendances externes
Fichiers de ressources
Fichiers d'en-tête
stdafx.h
targetver.h
Fichiers sources
ProjectMATLAB.cpp
stdafx.cpp
ReadMe.txt
Propriétés
main VCCodeFunction
C++
(Name) main
File c:\users\isoz\documents\visual
FullName main
IsDefault False
IsDelete False
IsFinal False
IsInjected False
IsInline False
IsOverloaded False
IsSealed False
IsTemplate False
TypeString int
C++
Prêt


ProjectMATLAB (Exécution) - Microsoft Visual Studio Express 2013 pour Windows Desktop
Espion 4
C:\Users\Isoz\Documents\Visual Studio 2013\ProjectMATLAB\ProjectMATLAB\ProjectMATLAB.exe
Information
MATLAB is there
OK
else {
MessageBox(0, "MATLAB is there", "Information", 0);
}
int Close();
std::string PlotCommand = "x=[0 1 2 3 4 5];y=[0 1 4 9 16 25];peaks";
engEvalString(matlabEngine, PlotCommand.c_str());
std::string QuitCommand = "pause(5);quit";
engEvalString(matlabEngine, QuitCommand.c_str());
}
Explorateur de solutions
Liste d'erreurs
Pile des appels
Prêt

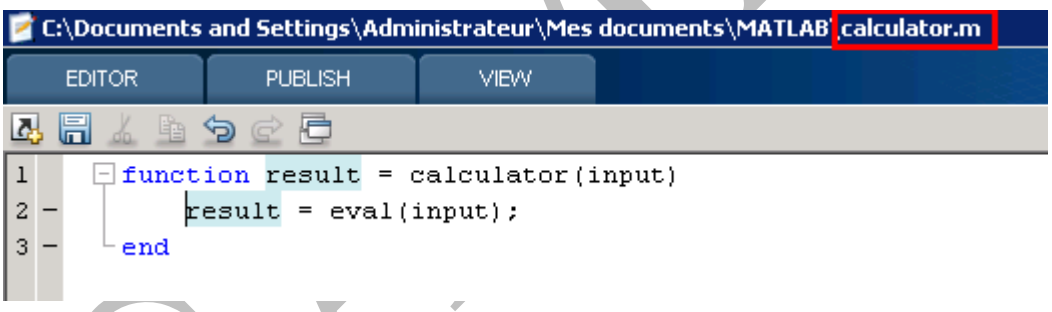

ProjectMATLAB (Exécution) - Microsoft Visual Studio Express 2013 pour Windows Desktop
C:\Users\Isoz\Documents\Visual Studio 2013\ProjectMATLAB\ProjectMATLAB\ProjectMATLAB.exe
Figure 1
File Edit View Insert Tools Desktop Window Help
Peaks
else {
MessageBox(0, "MATLAB is there", "I
}
int Close();
std::string PlotCommand = "x=[0 1 2 3 4
engEvalString(matlabEngine, PlotCommand
std::string QuitCommand = "pause(5);qui
engEvalString(matlabEngine, QuitCommand
}
Liste d'erreurs
Pile des appels
Prêt
Ln 24
Col 43
Car 37
INS

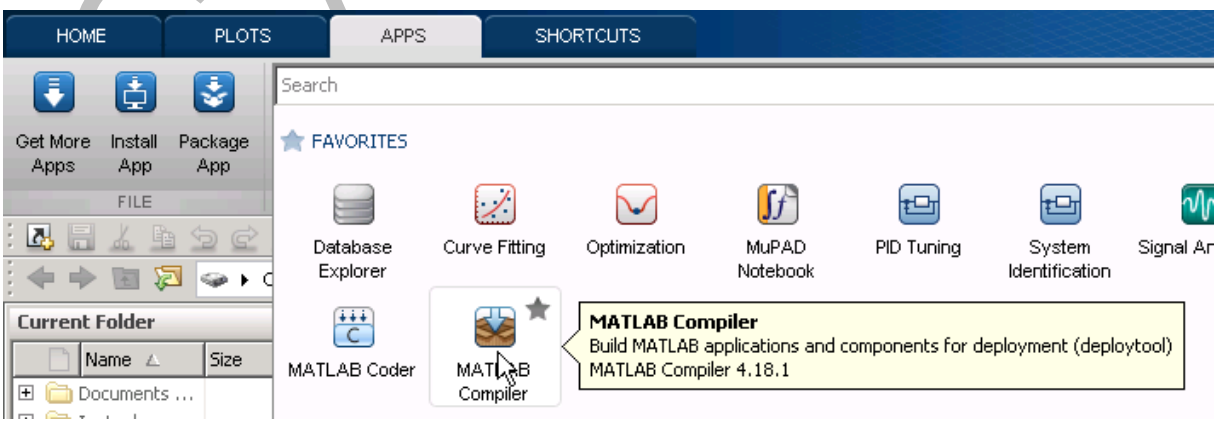

ProjectMATLAB - Microsoft Visual Studio Express 2013 pour Windows Desktop
FICHIER MODIFIER AFFICHER PROJET GÉNÉRER DÉBOGUER ÉQUIPE OUTILS TEST FENÊTRE AIDE
Connexion
Débogueur Windows local
Debug
x64
ProjectMATLAB.cpp
ProjectMATLAB
(Portée globale)
main()
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <engine.h>
#include <windows.h>
#pragma comment (lib,"libmat.lib")
#pragma comment (lib,"libmx.lib")
#pragma comment (lib,"libmex.lib")
#pragma comment (lib,"libeng.lib")
using namespace std;
int main()
{
void * vpDcom = NULL;
int iReturnValue;
Engine* matlabEngine = engOpenSingleUse(0, vpDcom, &iReturnValue);
double *data;
int i;
mxArray *mtlbDataArray;
// code to fill in values of x
mtlbDataArray = mxCreateDoubleMatrix(5, 5, mxREAL);
data = mxGetPr(mtlbDataArray);
for (i = 0; i<25; i++){
data[i] = 1;
}
engPutVariable(matlabEngine, "myArray", mtlbDataArray);
mxDestroyArray(mtlbDataArray);
}
100 %
Explorateur de solutions
Rechercher dans l'Explorateur de solutions (Ctrl+è)
Solution 'ProjectMATLAB' (1 projet)
ProjectMATLAB
Dépendances externes
Fichiers de ressources
Fichiers d'en-tête
stdafx.h
targetver.h
Fichiers sources
ProjectMATLAB.cpp
stdafx.cpp
ReadMe.txt
Propriétés
main VCCodeFunction
C++
(Name) main
File c:\users\isoz\documents\visual
FullName main
IsDefault False
IsDelete False
IsFinal False
IsInjected False
IsInline False
IsOverloaded False
IsSealed False
IsTemplate False
TypeString int
C++
Liste d'erreurs
0 erreurs
0 avertissements
0 messages
Liste d'erreurs de recherche
Description Fichier Ligne Colo... Projet
Liste d'erreurs Sortie
Prêt

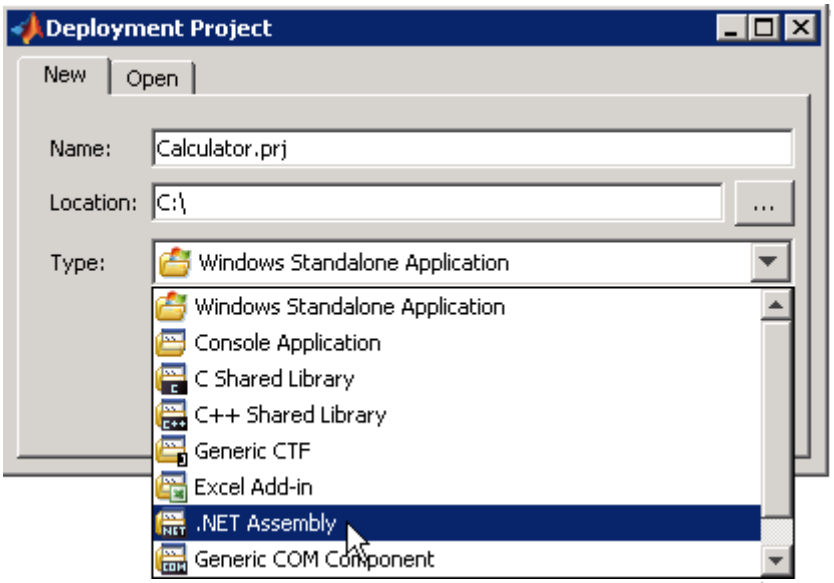


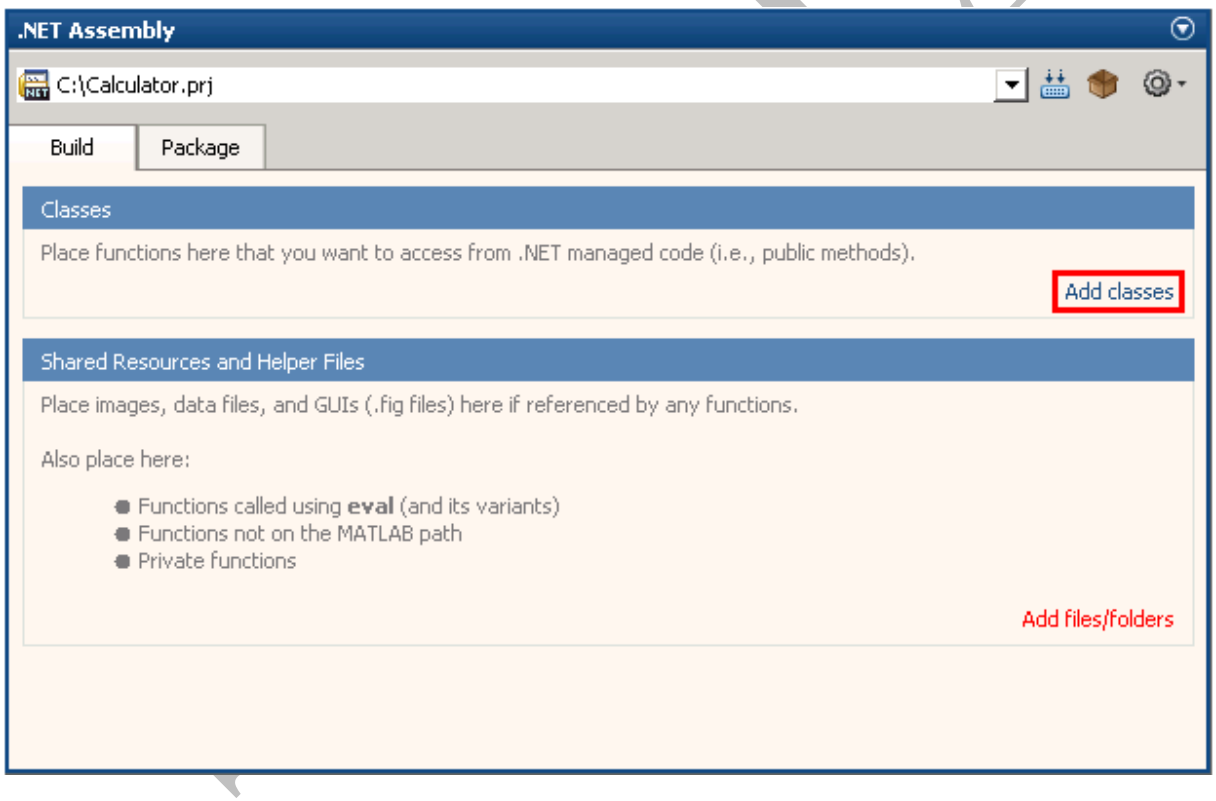
Compile DLL and call from C#

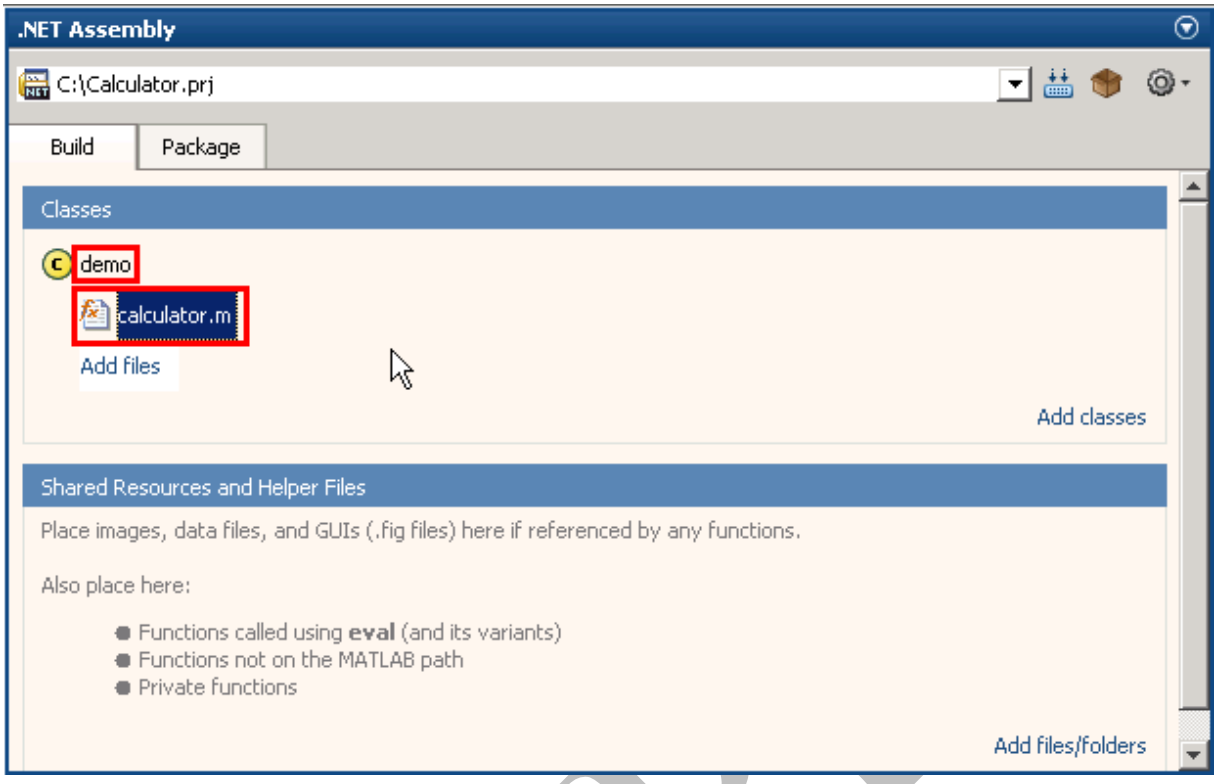
Passing a scalar

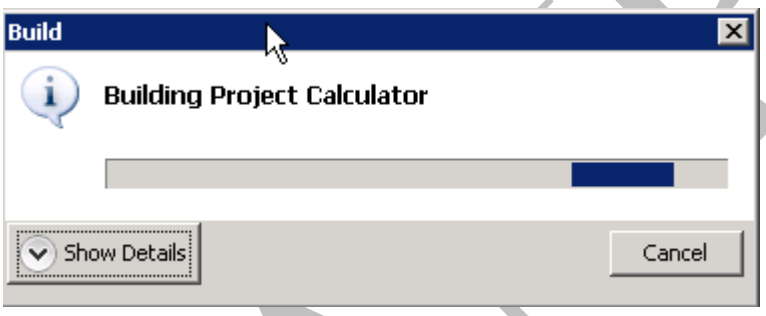





Deployment Project
New
Open
Name:
Calculator.prj
Location:
C:\
...
Type:
Windows Standalone Application
Windows Standalone Application
Console Application
C Shared Library
C++ Shared Library
Generic CTF
Excel Add-in
.NET Assembly
Generic COM Component

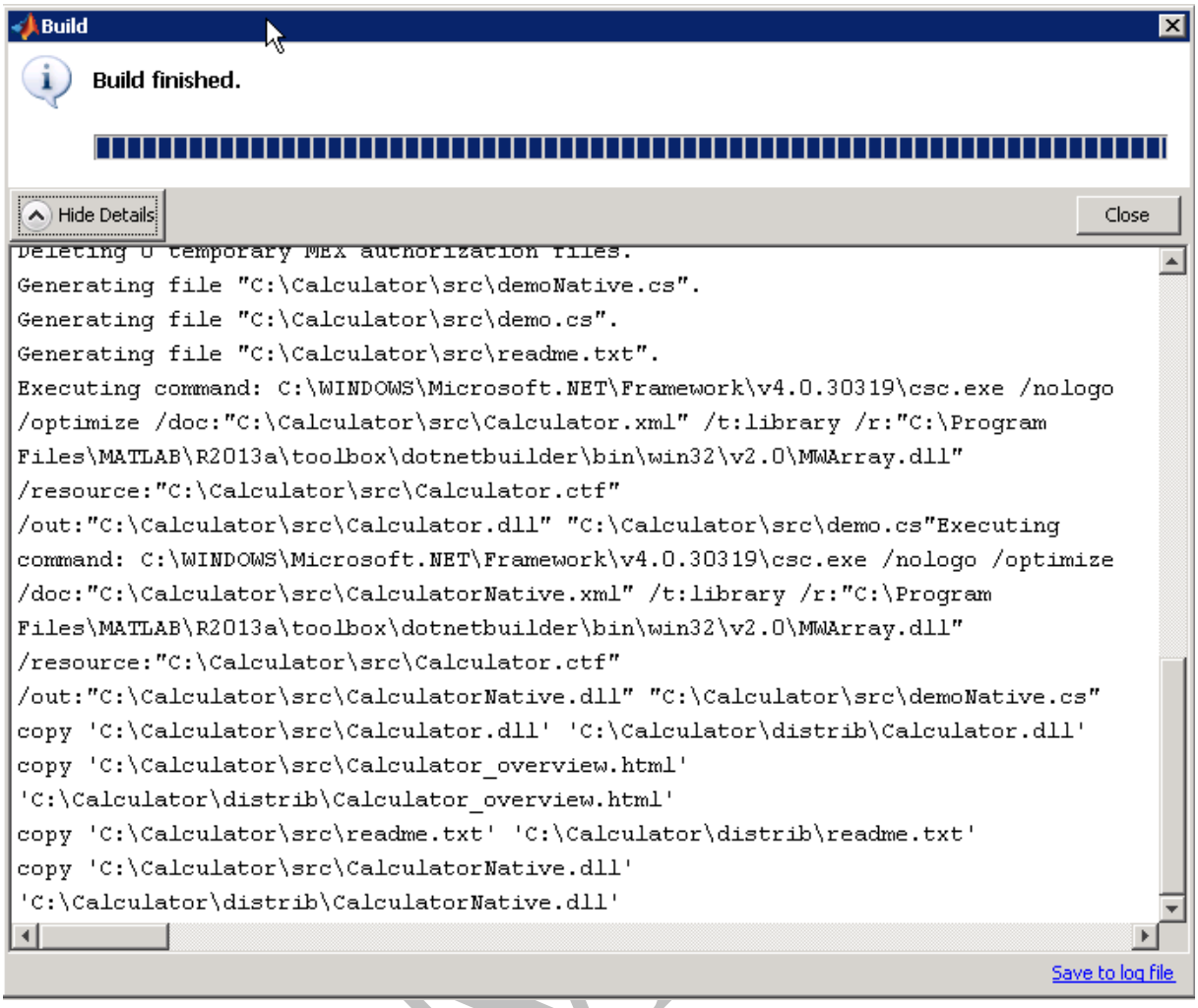

.NET Assembly
C:\Calculator.prj
Build
Package
Classes
Place functions here that you want to access from .NET managed code (i.e., public methods).
Add classes
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if referenced by any functions.
Also place here:
Functions called using eval (and its variants)
Functions not on the MATLAB path
Private functions
Add files/folders



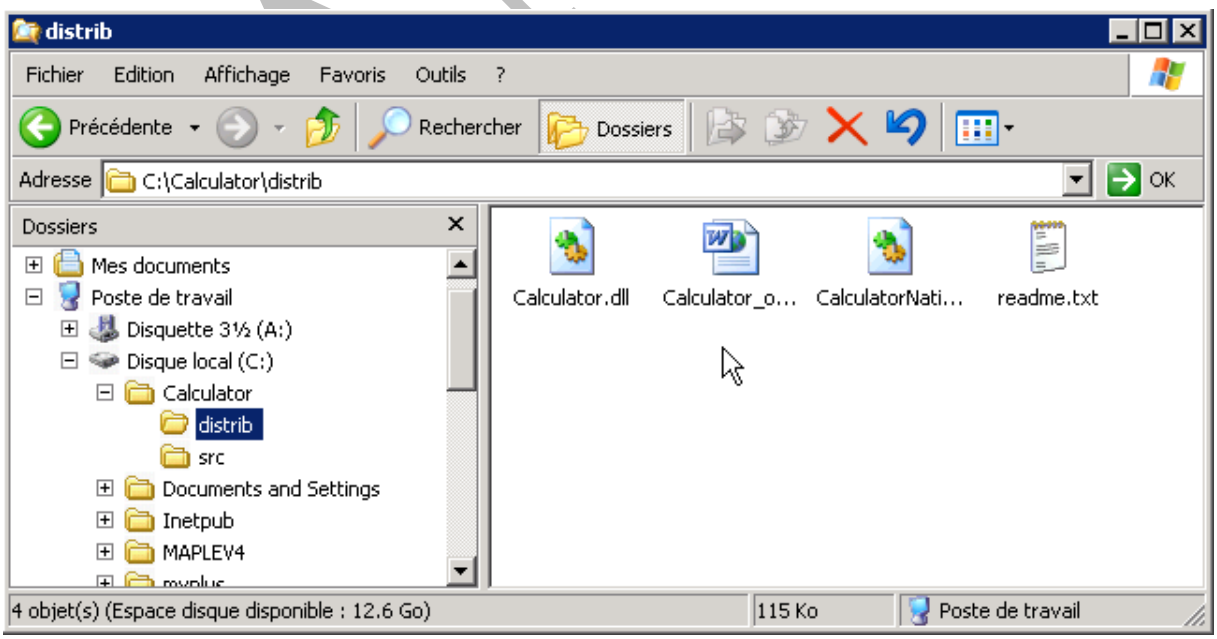


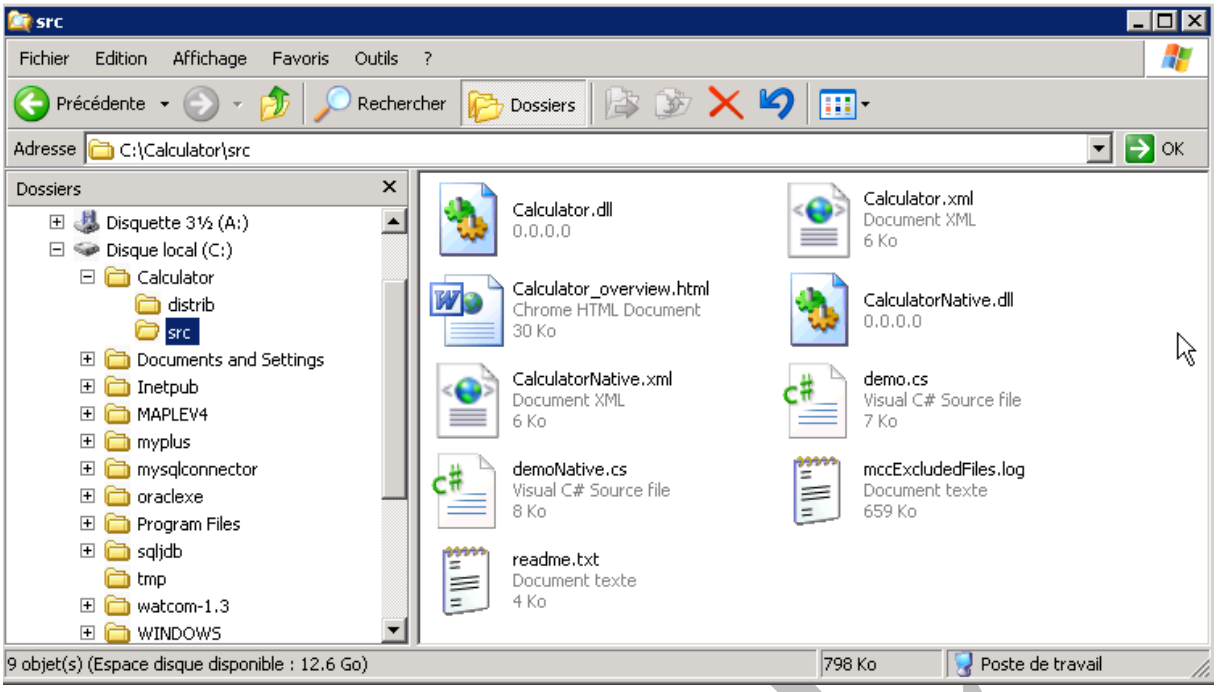
After a long time...

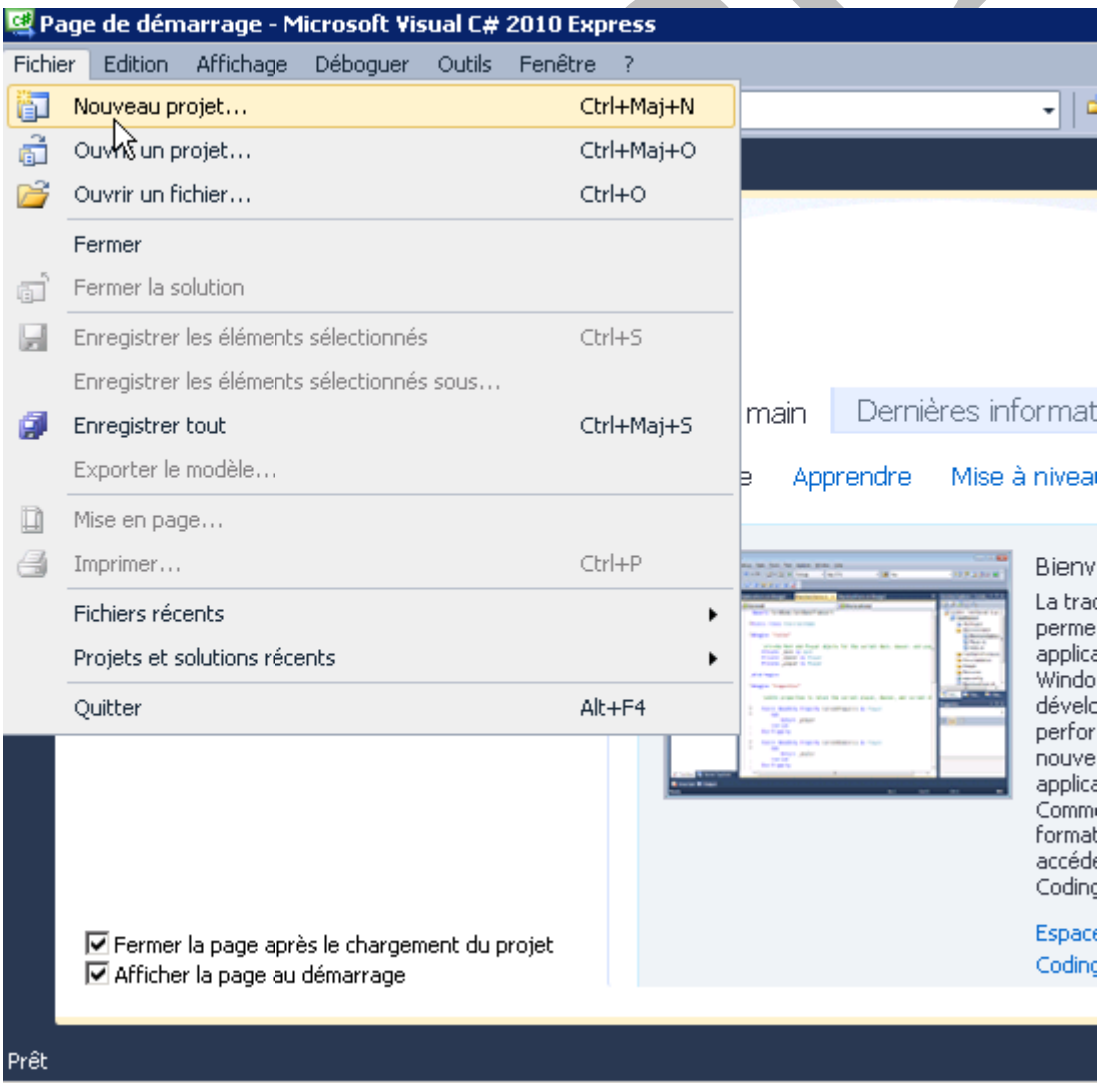
SAMPLE

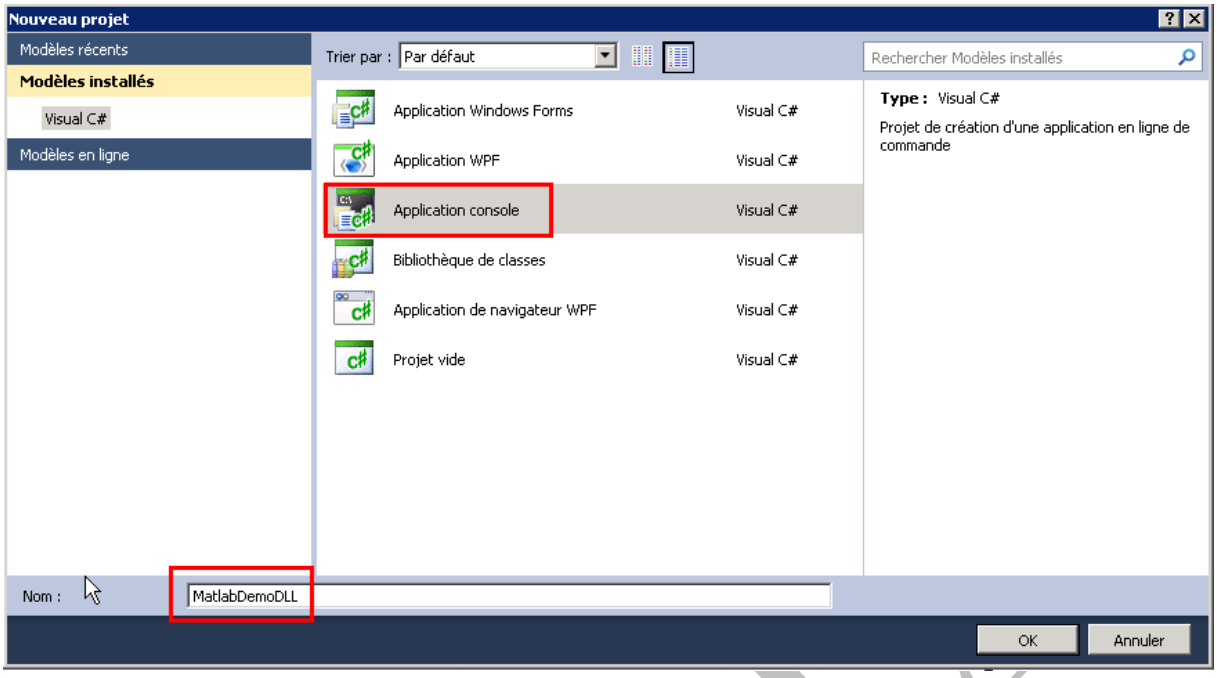

Build
Build finished.
Hide Details
Close
Deleting 0 temporary MEX authorization files.
Generating file "C:\Calculator\src\demoNative.cs".
Generating file "C:\Calculator\src\demo.cs".
Generating file "C:\Calculator\src\readme.txt".
Executing command: C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\csc.exe /nologo
/optimize /doc:"C:\Calculator\src\Calculator.xml" /t:library /r:"C:\Program
Files\MATLAB\R2013a\toolbox\dotnetbuilder\bin\win32\v2.0\MWArray.dll"
/resource:"C:\Calculator\src\Calculator.ctf"
/out:"C:\Calculator\src\Calculator.dll" "C:\Calculator\src\demo.cs"Executing
command: C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\csc.exe /nologo /optimize
/doc:"C:\Calculator\src\CalculatorNative.xml" /t:library /r:"C:\Program
Files\MATLAB\R2013a\toolbox\dotnetbuilder\bin\win32\v2.0\MWArray.dll"
/resource:"C:\Calculator\src\Calculator.ctf"
/out:"C:\Calculator\src\CalculatorNative.dll" "C:\Calculator\src\demoNative.cs"
copy 'C:\Calculator\src\Calculator.dll' 'C:\Calculator\distrib\Calculator.dll'
copy 'C:\Calculator\src\Calculator_overview.html'
'C:\Calculator\distrib\Calculator_overview.html'
copy 'C:\Calculator\src\readme.txt' 'C:\Calculator\distrib\readme.txt'
copy 'C:\Calculator\src\CalculatorNative.dll'
'C:\Calculator\distrib\CalculatorNative.dll'
Save to log file

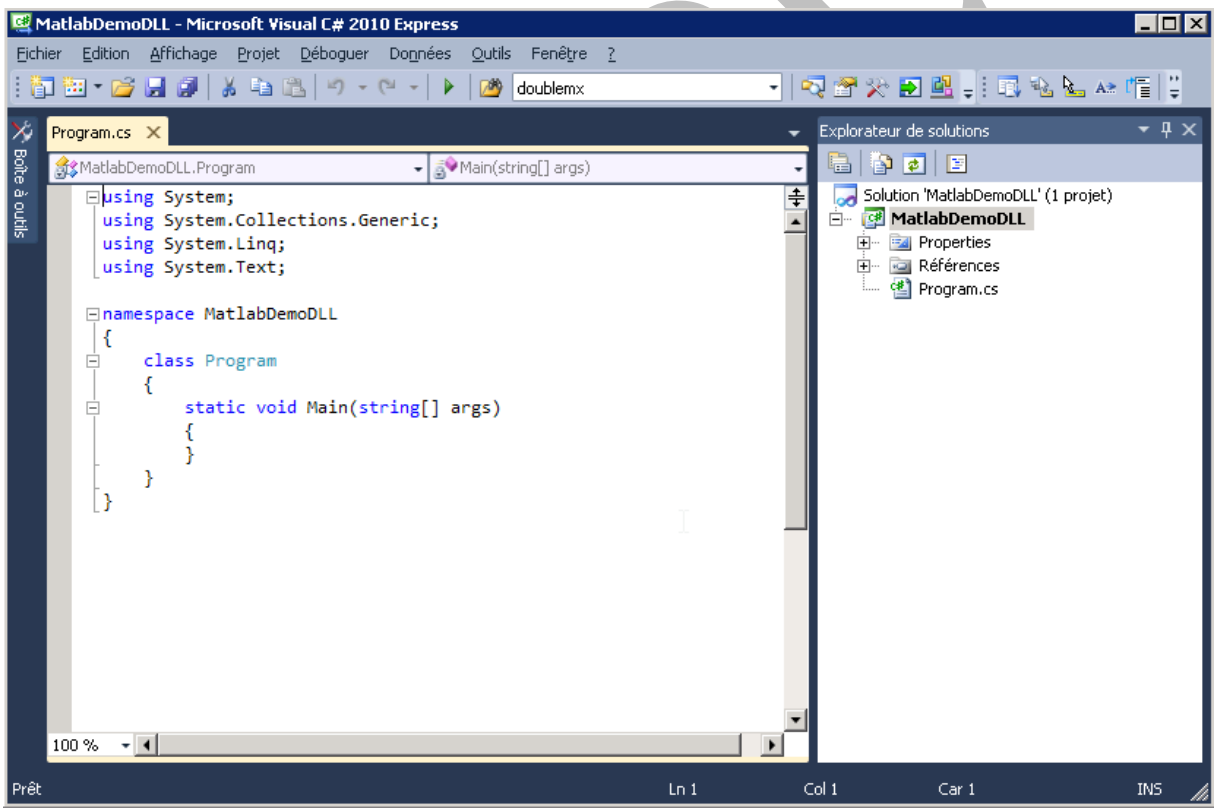
You get:

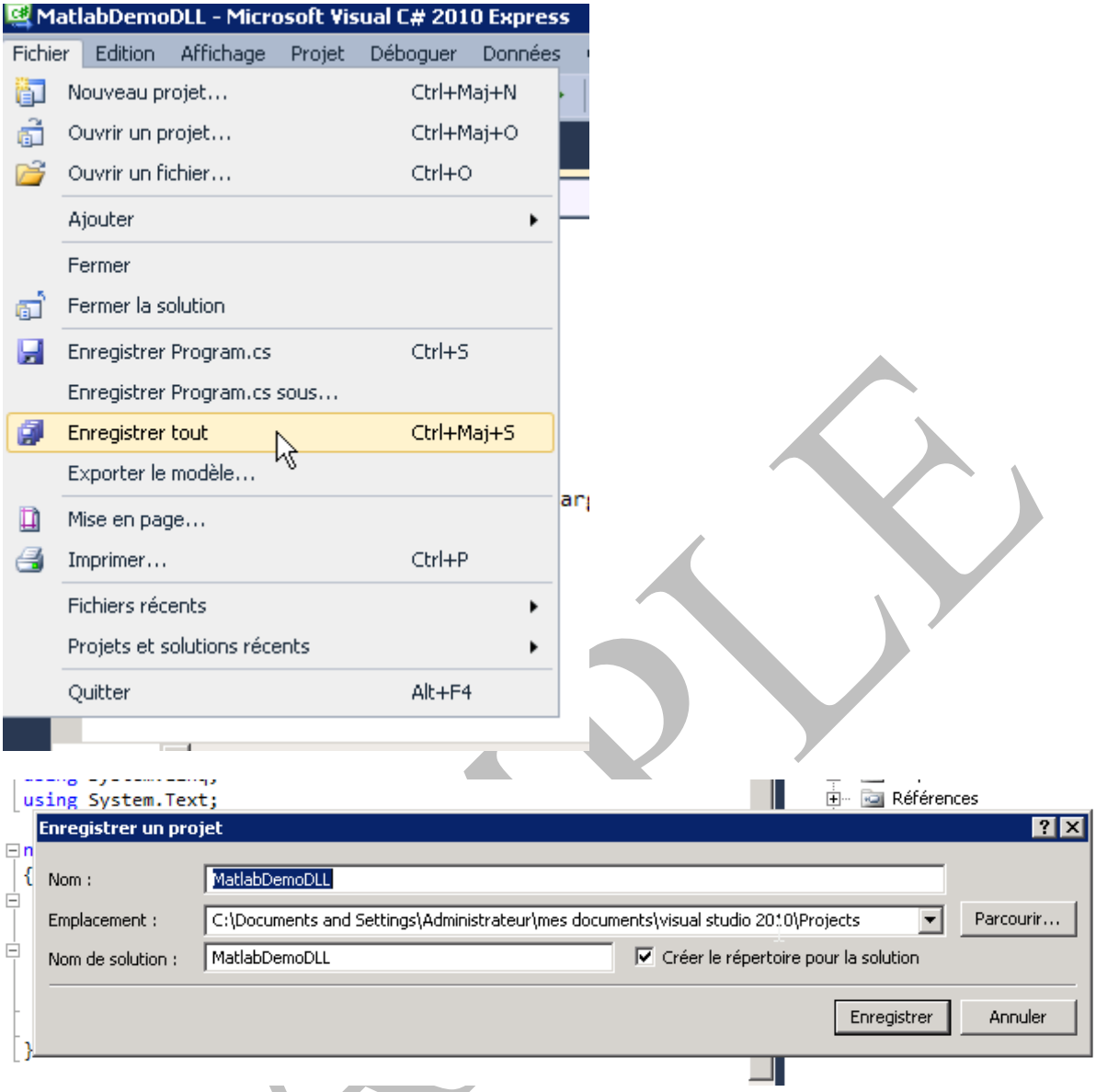

distrib
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Dossiers
Adresse C:\Calculator\distrib
OK
Dossiers
Mes documents
Poste de travail
Disquette 3½ (A:)
Disque local (C:)
Calculator
distrib
src
Documents and Settings
Inetpub
MAPLEV4
Calculator.dll
Calculator_o...
CalculatorNati...
readme.txt
4 objet(s) (Espace disque disponible : 12.6 Go)
115 Ko
Poste de travail


src
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Dossiers
Adresse C:\Calculator\src
OK
Dossiers
Disquette 3½ (A:)
Disque local (C:)
Calculator
distrib
src
Documents and Settings
Inetpub
MAPLEV4
myplus
mysqlconnector
oraclexe
Program Files
sqljdb
tmp
watcom-1.3
WINDOWS
Calculator.dll
0.0.0.0
Calculator.xml
Document XML
6 Ko
Calculator_overview.html
Chrome HTML Document
30 Ko
CalculatorNative.dll
0.0.0.0
CalculatorNative.xml
Document XML
6 Ko
demo.cs
Visual C# Source file
7 Ko
demoNative.cs
Visual C# Source file
8 Ko
mccExcludedFiles.log
Document texte
659 Ko
readme.txt
Document texte
4 Ko
9 objet(s) (Espace disque disponible : 12.6 Go)
798 Ko
Poste de travail

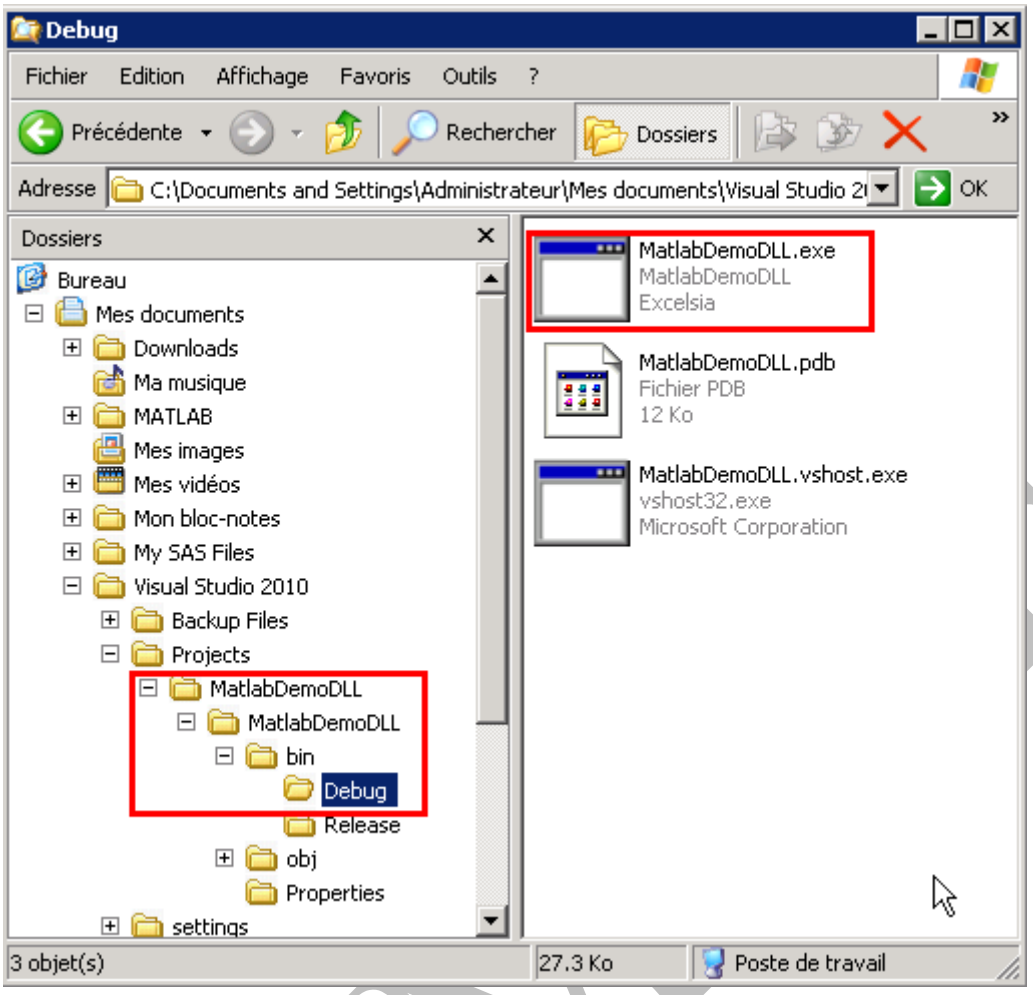

Page de démarrage - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Déboguer Outils Fenêtre ?
Nouveau projet... Ctrl+Maj+N
Ouvrir un projet... Ctrl+Maj+O
Ouvrir un fichier... Ctrl+O
Fermer
Fermer la solution
Enregistrer les éléments sélectionnés Ctrl+S
Enregistrer les éléments sélectionnés sous...
Enregistrer tout Ctrl+Maj+S
Exporter le modèle...
Mise en page...
Imprimer... Ctrl+P
Fichiers récents
Projets et solutions récents
Quitter Alt+F4
Dernières informat
Apprendre
Fermer la page après le chargement du projet
Afficher la page au démarrage
Prêt


Nouveau projet
Modèles récents
Modèles installés
Visual C#
Modèles en ligne
Trier par : Par défaut
Rechercher Modèles installés
Application Windows Forms Visual C#
Application WPF Visual C#
Application console Visual C#
Bibliothèque de classes Visual C#
Application de navigateur WPF Visual C#
Projet vide Visual C#
Type : Visual C#
Projet de création d'une application en ligne de commande
Nom :
MatlabDemoDLL
OK
Annuler


MatlabDemoDLL - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données Outils Fenêtre ?
doublemx
Program.cs
MatlabDemoDLL.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MatlabDemoDLL
{
class Program
{
static void Main(string[] args)
{
}
}
}
Explorateur de solutions
Solution 'MatlabDemoDLL' (1 projet)
MatlabDemoDLL
Properties
Références
Program.cs
Boîte à outils
100 %
Prêt
Ln 1
Col 1
Car 1
INS

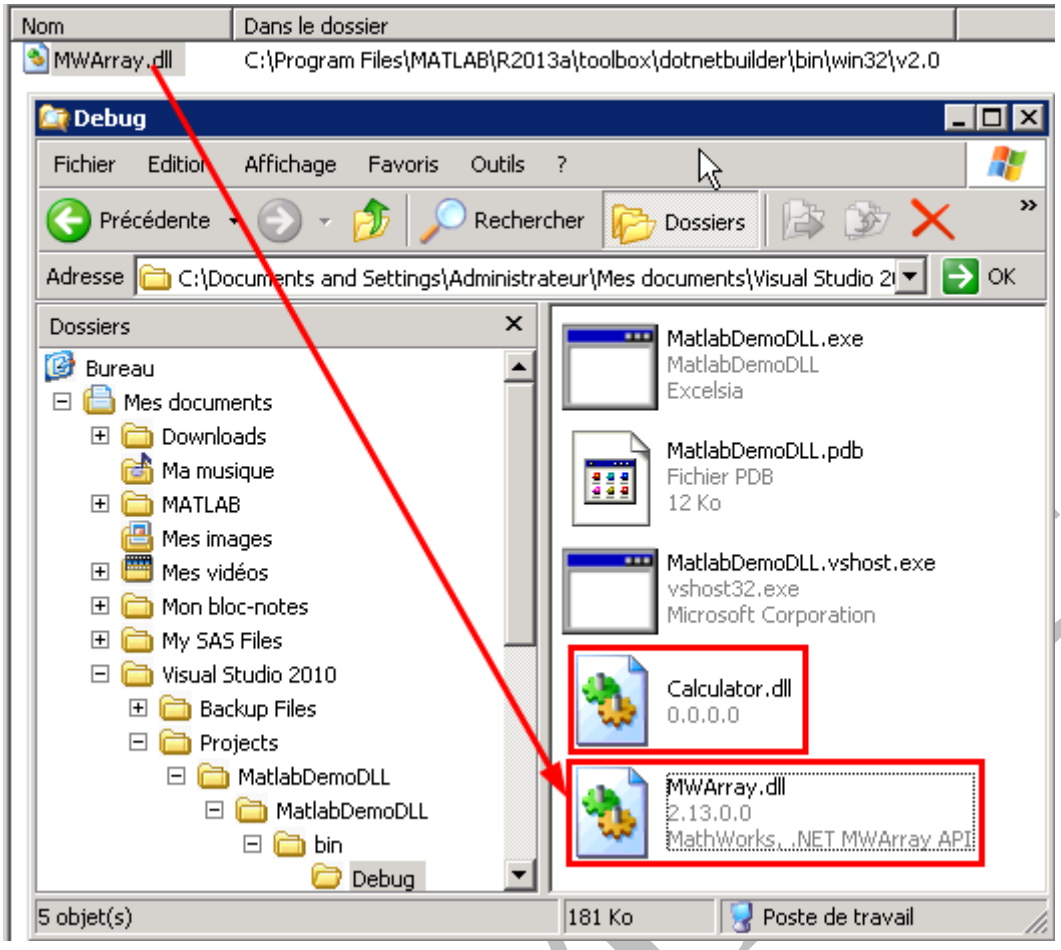

MatlabDemoDLL - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données
Nouveau projet... Ctrl+Maj+N
Ouvrir un projet... Ctrl+Maj+O
Ouvrir un fichier... Ctrl+O
Ajouter
Fermer
Fermer la solution
Enregistrer Program.cs Ctrl+S
Enregistrer Program.cs sous...
Enregistrer tout Ctrl+Maj+S
Exporter le modèle...
Mise en page...
Imprimer... Ctrl+P
Fichiers récents
Projets et solutions récents
Quitter Alt+F4
using System.Text;
Références
Enregistrer un projet
Nom : MatlabDemoDLL
Emplacement : C:\Documents and Settings\Administrateur\mes documents\visual studio 2010\Projects
Parcourir...
Nom de solution : MatlabDemoDLL
Créer le répertoire pour la solution
Enregistrer
Annuler

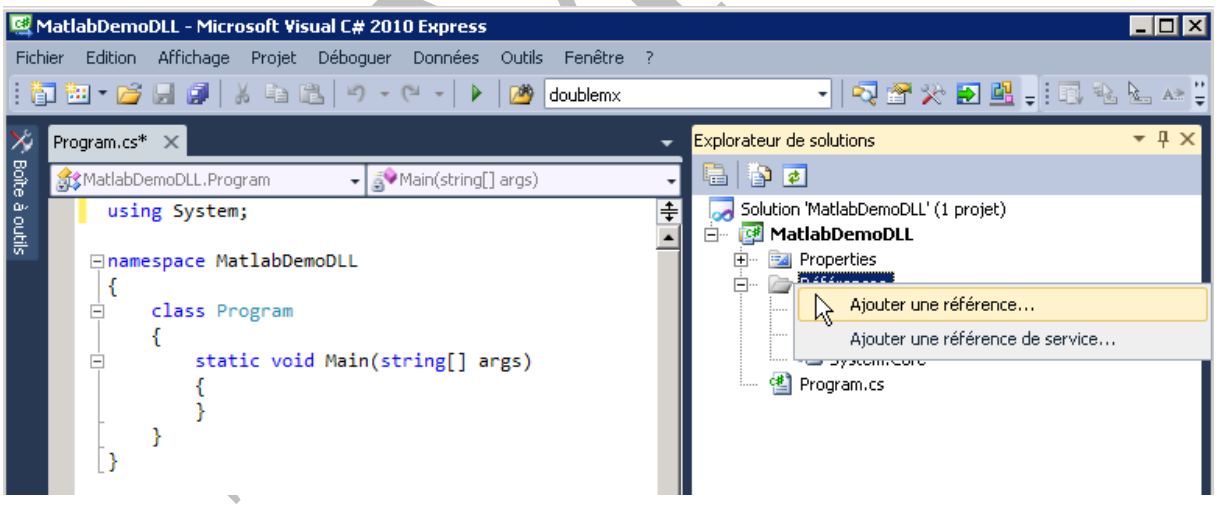
SAMPLE

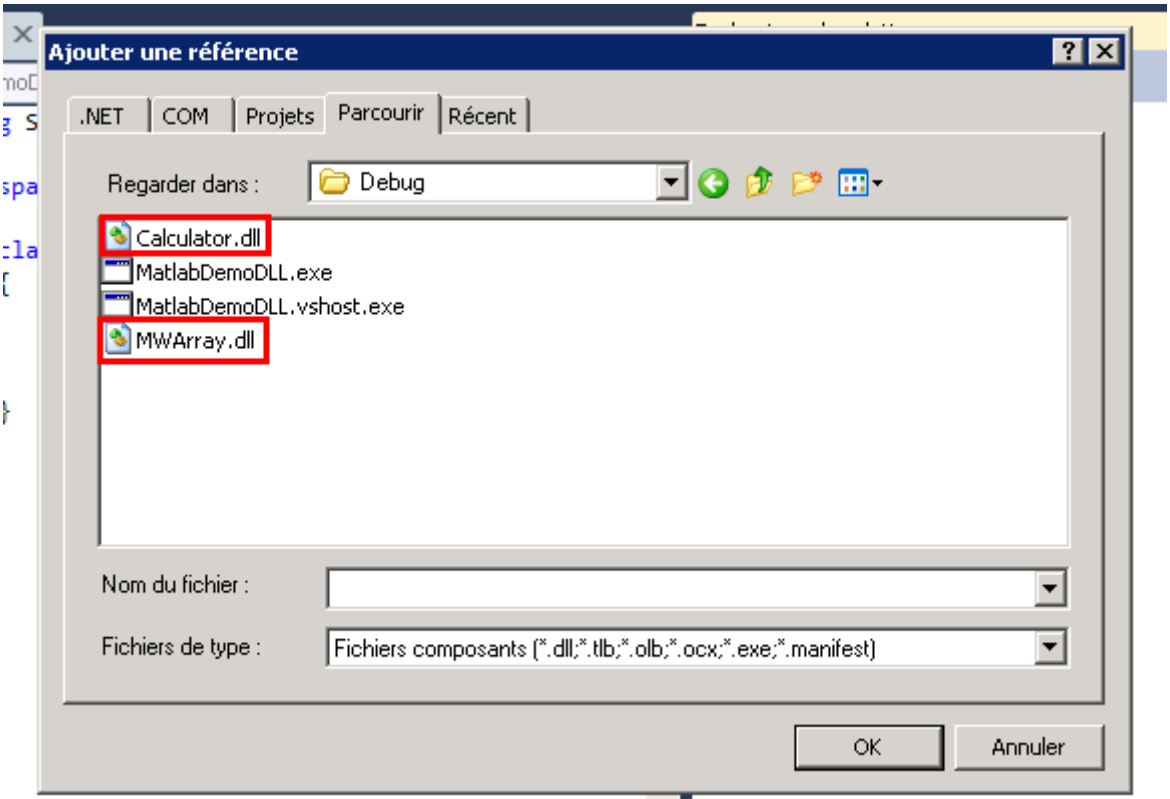


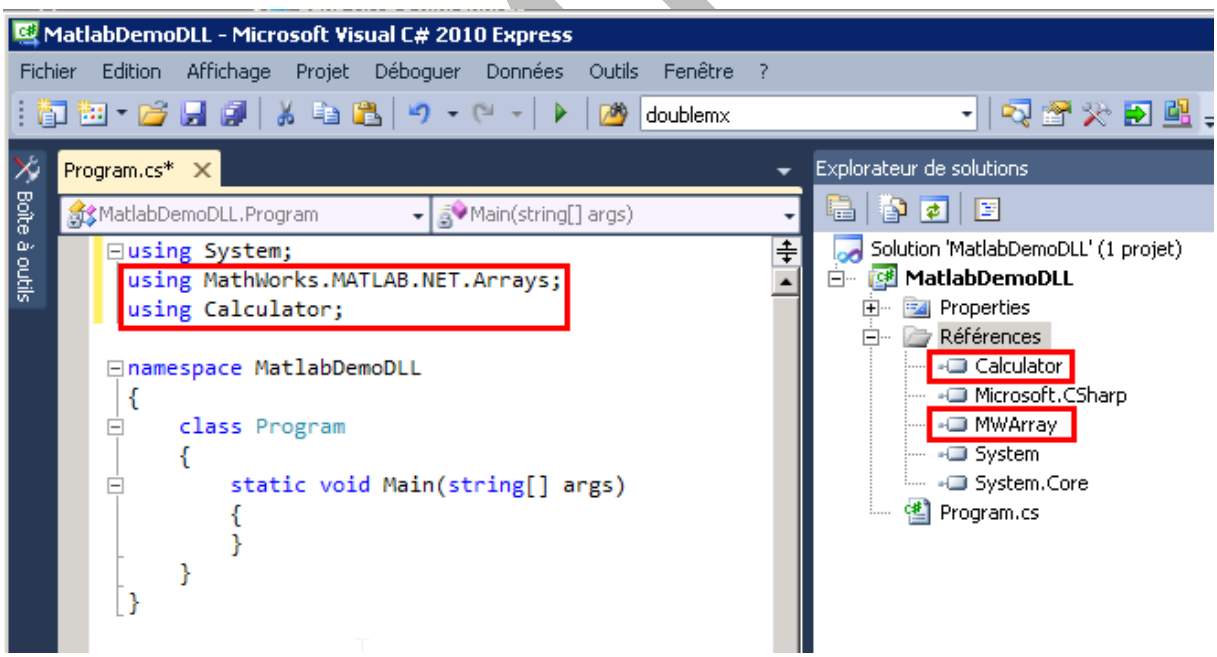
Note that, in order to call the generated DLL, the DLL must reside either in the same directory as your executable or in a directory on your system path through the environment variables of the system!

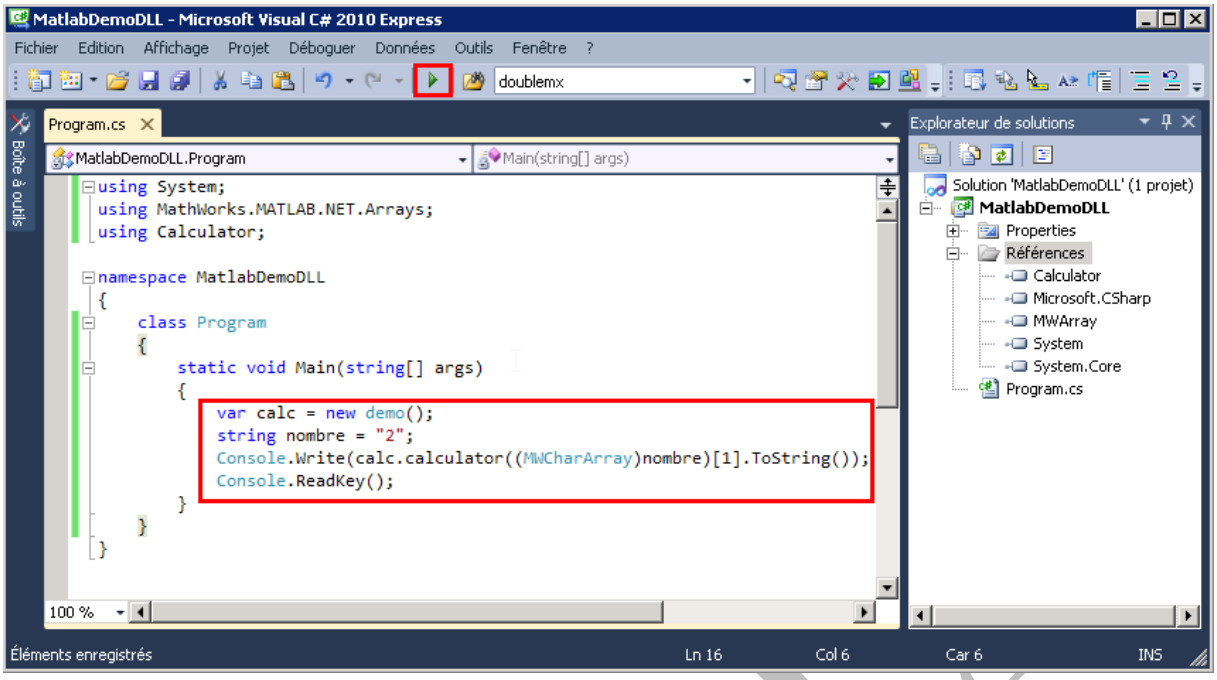
SAMPLE

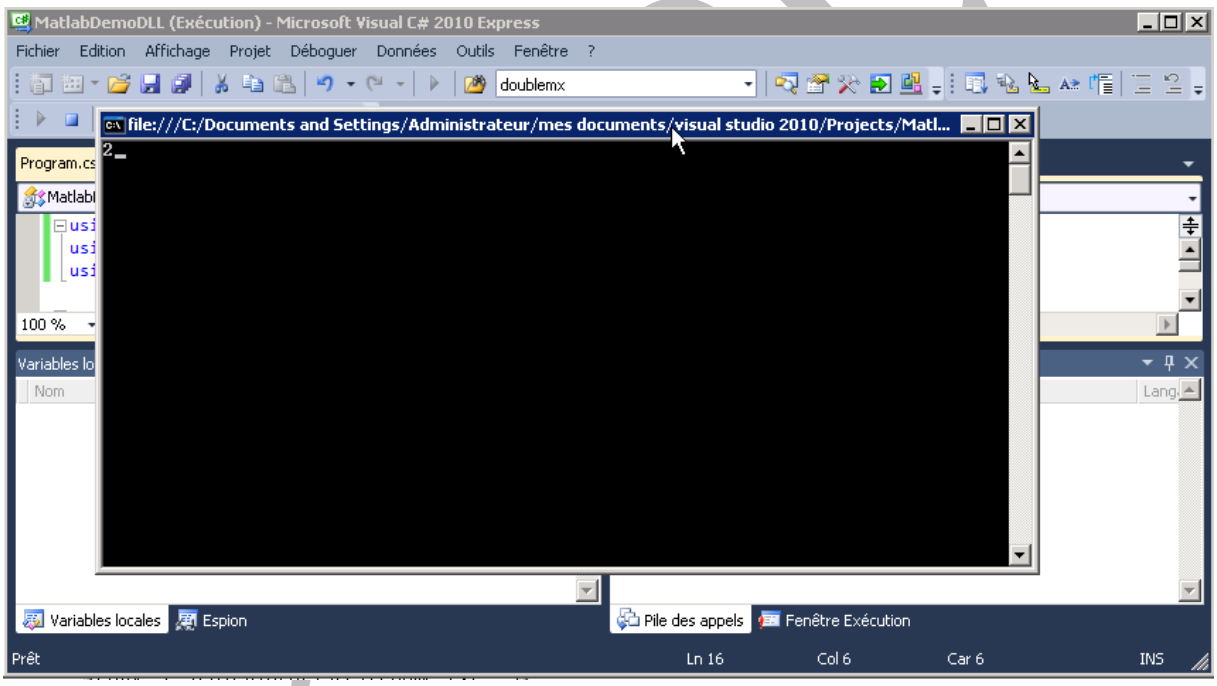

Nom
Dans le dossier
MWArray.dll
C:\Program Files\MATLAB\R2013a\toolbox\dotnetbuilder\bin\win32\v2.0
Debug
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Dossiers
Adresse
C:\Documents and Settings\Administrateur\Mes documents\Visual Studio 2
OK
Dossiers
Bureau
Mes documents
Downloads
Ma musique
MATLAB
Mes images
Mes vidéos
Mon bloc-notes
My SAS Files
Visual Studio 2010
Backup Files
Projects
MatlabDemoDLL
MatlabDemoDLL
bin
Debug
MatlabDemoDLL.exe
MatlabDemoDLL
Excelsia
MatlabDemoDLL.pdb
Fichier PDB
12 Ko
MatlabDemoDLL.vshost.exe
vshost32.exe
Microsoft Corporation
Calculator.dll
0.0.0.0
MWArray.dll
2.13.0.0
MathWorks, .NET MWArray API
5 objet(s)
181 Ko
Poste de travail


MatlabDemoDLL - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données Outils Fenêtre ?
doublemx
Program.cs*
MatlabDemoDLL.Program
Main(string[] args)
using System;
namespace MatlabDemoDLL
{
class Program
{
static void Main(string[] args)
{
}
}
}
Boîte à outils
Explorateur de solutions
Solution 'MatlabDemoDLL' (1 projet)
MatlabDemoDLL
Properties
Ajouter une référence...
Ajouter une référence de service...
Program.cs


Ajouter une référence
.NET
COM
Projets
Parcourir
Récent
Regarder dans :
Debug
Calculator.dll
MatlabDemoDLL.exe
MatlabDemoDLL.vshost.exe
MWArray.dll
Nom du fichier :
Fichiers de type :
Fichiers composants (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)
OK
Annuler

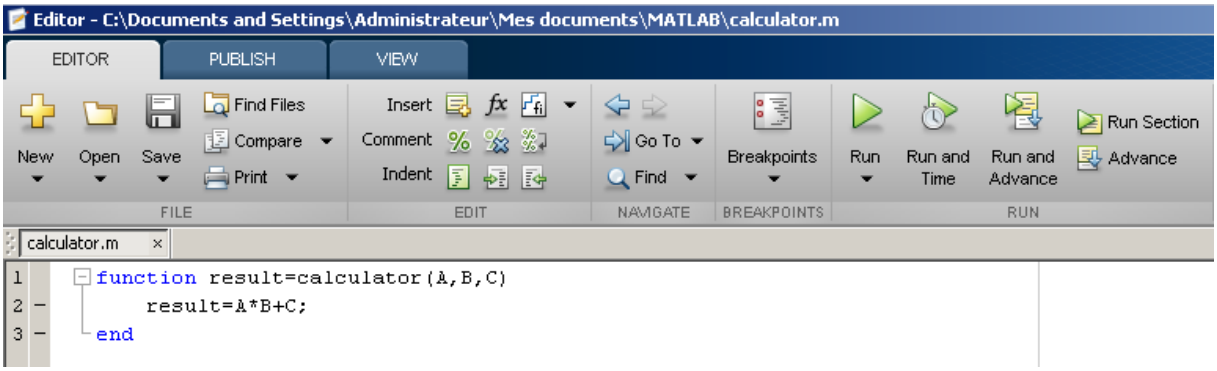

MatlabDemoDLL - Microsoft Visual C# 2010 Express
Fichier Edition Affichage Projet Déboguer Données Outils Fenêtre ?
doublemx
Program.cs*
MatlabDemoDLL.Program
Main(string[] args)
using System;
using MathWorks.MATLAB.NET.Arrays;
using Calculator;
namespace MatlabDemoDLL
{
class Program
{
static void Main(string[] args)
{
}
}
}
Explorateur de solutions
Solution 'MatlabDemoDLL' (1 projet)
MatlabDemoDLL
Properties
Références
Calculator
Microsoft.CSharp
MWArray
System
System.Core
Program.cs



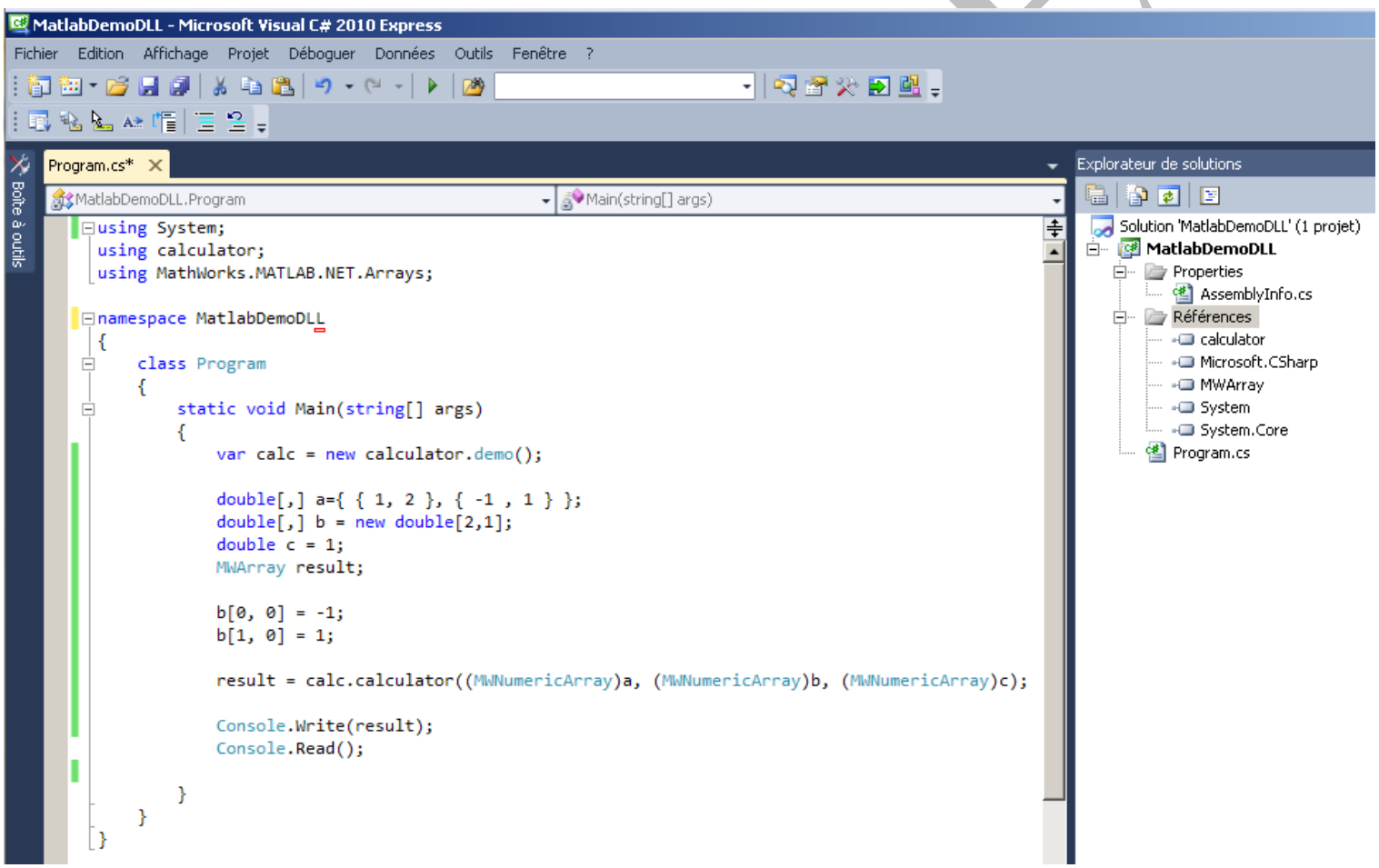


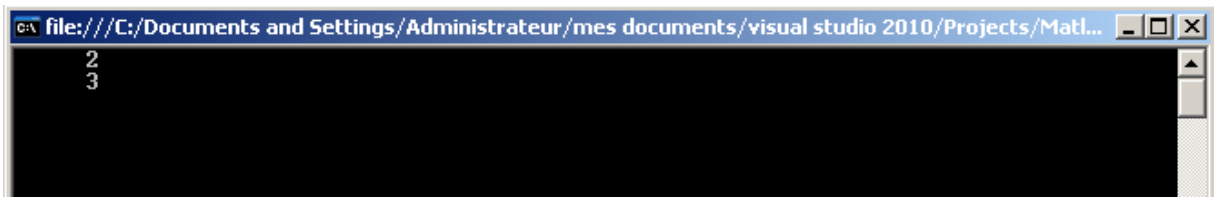
Passing a constant, vector and matrix

Because C# vectors are column vectors and MATLAB vectors are line vectors, there is a trap for passing vector variables.

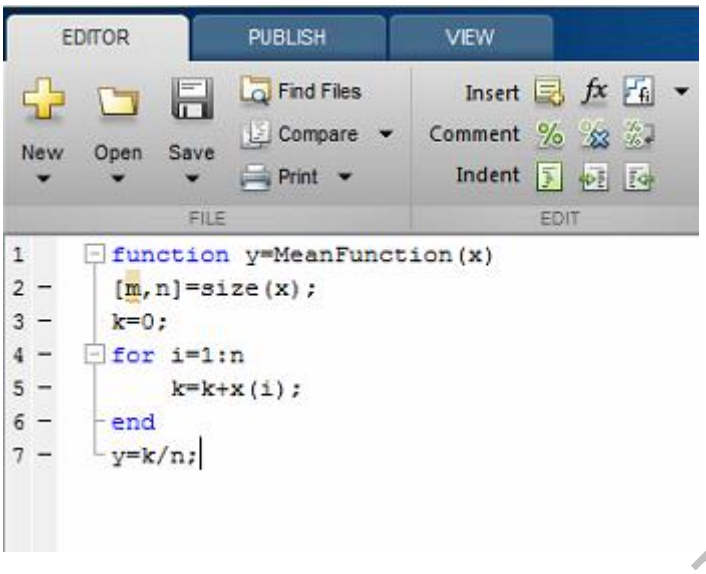


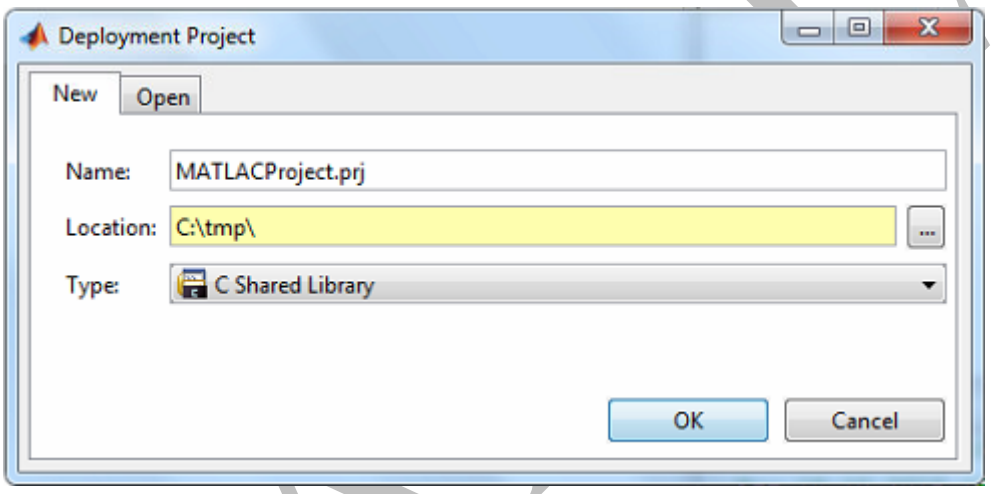
$$A=\begin{pmatrix}1 & 2\\ -1 & 1\end{pmatrix}, B=\begin{pmatrix}-1\\ 1\end{pmatrix}, C=1$$



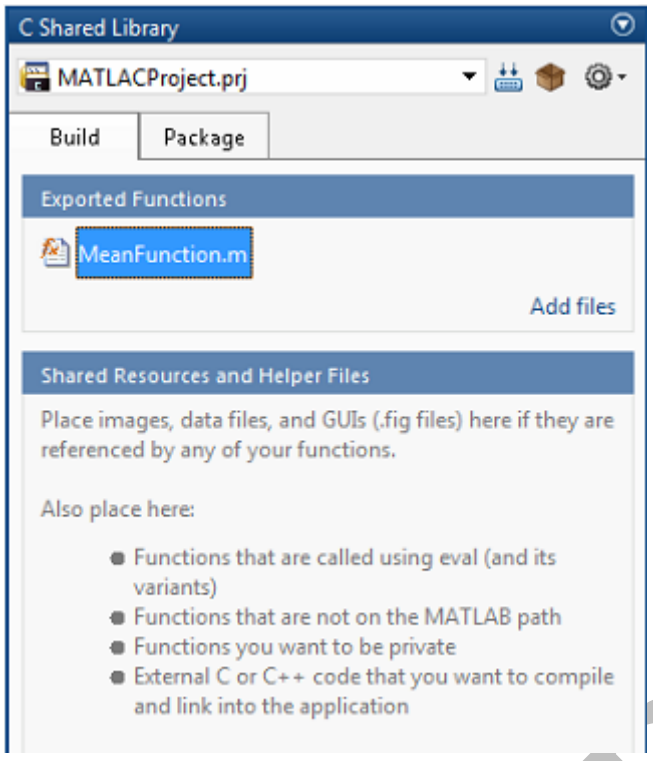


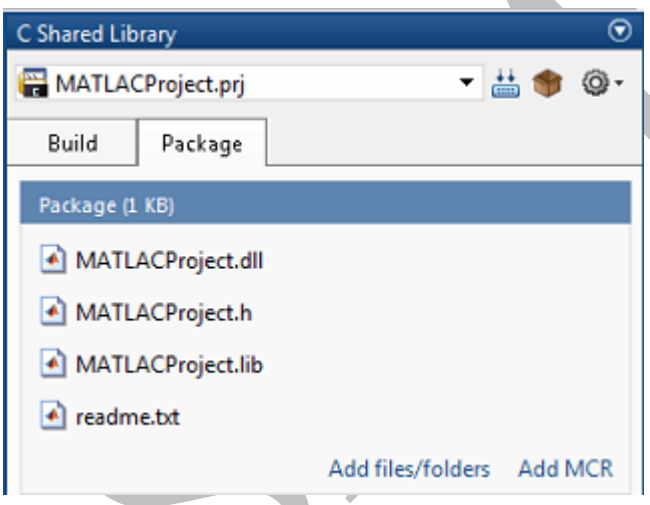
Compile DLL and call from C



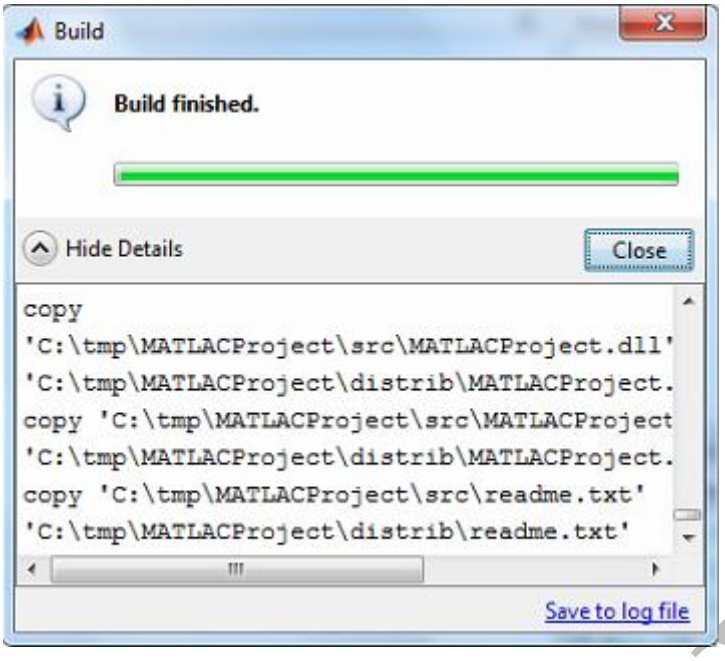


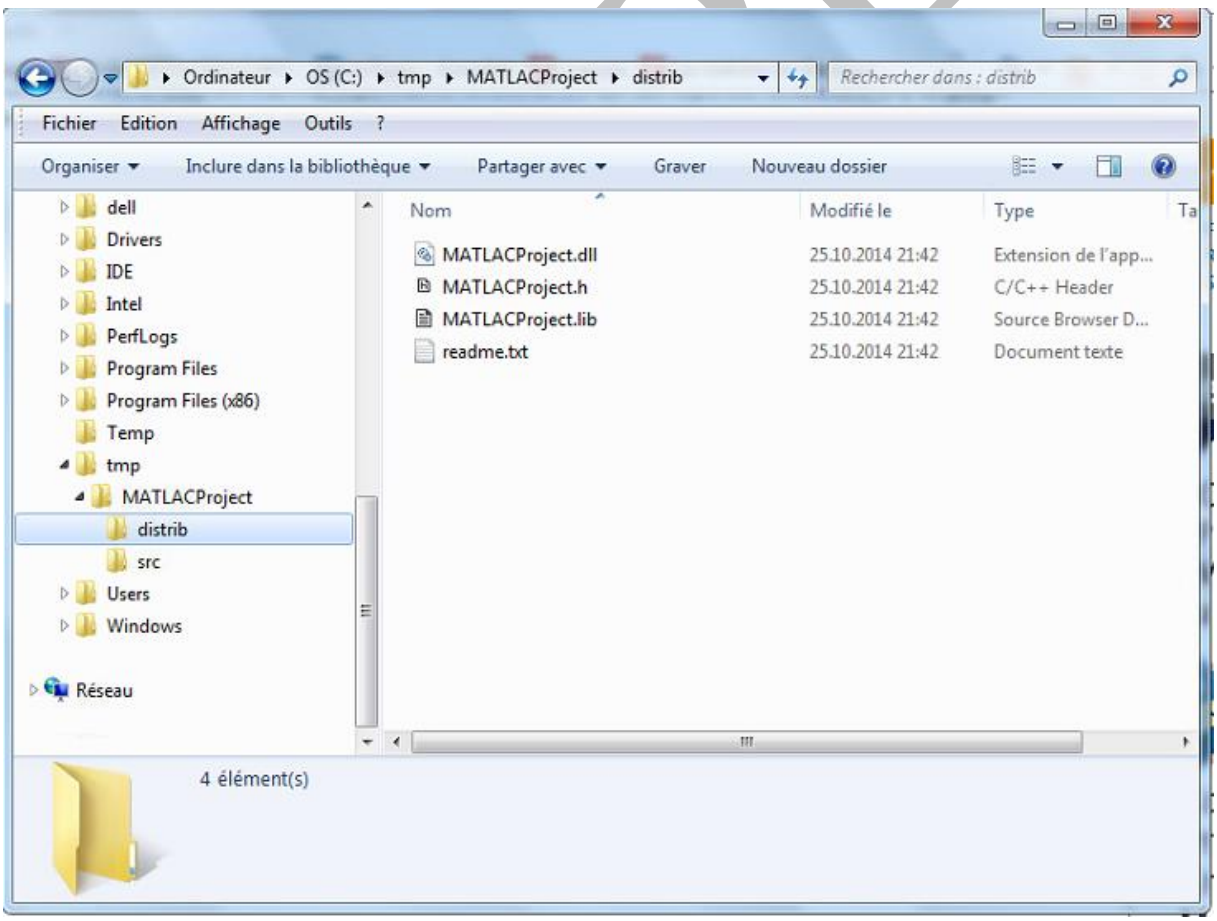
SAMPLE

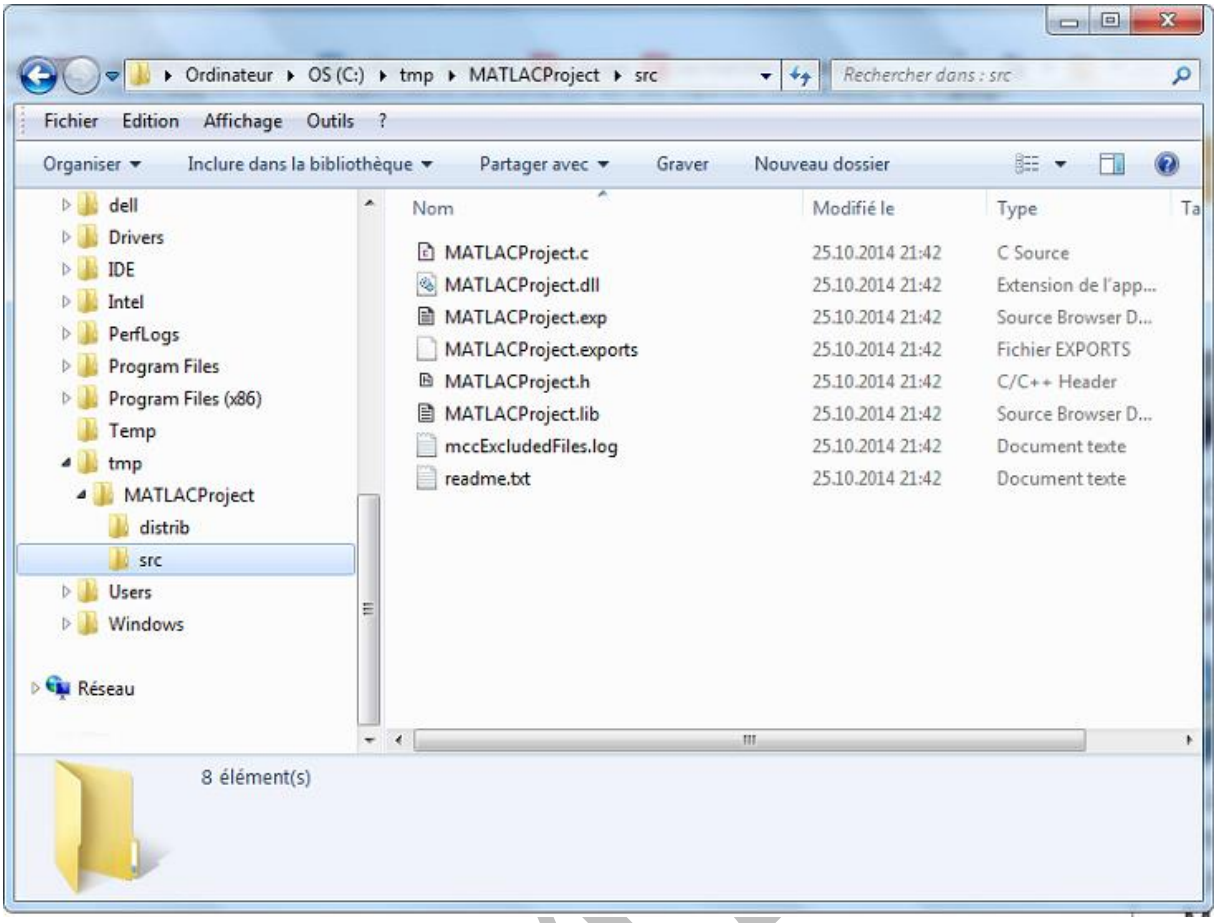

C Shared Library
MATLACProject.prj
Build
Package
Exported Functions
MeanFunction.m
Add files
Shared Resources and Helper Files
Place images, data files, and GUIs (.fig files) here if they are referenced by any of your functions.
Also place here:
Functions that are called using eval (and its variants)
Functions that are not on the MATLAB path
Functions you want to be private
External C or C++ code that you want to compile and link into the application

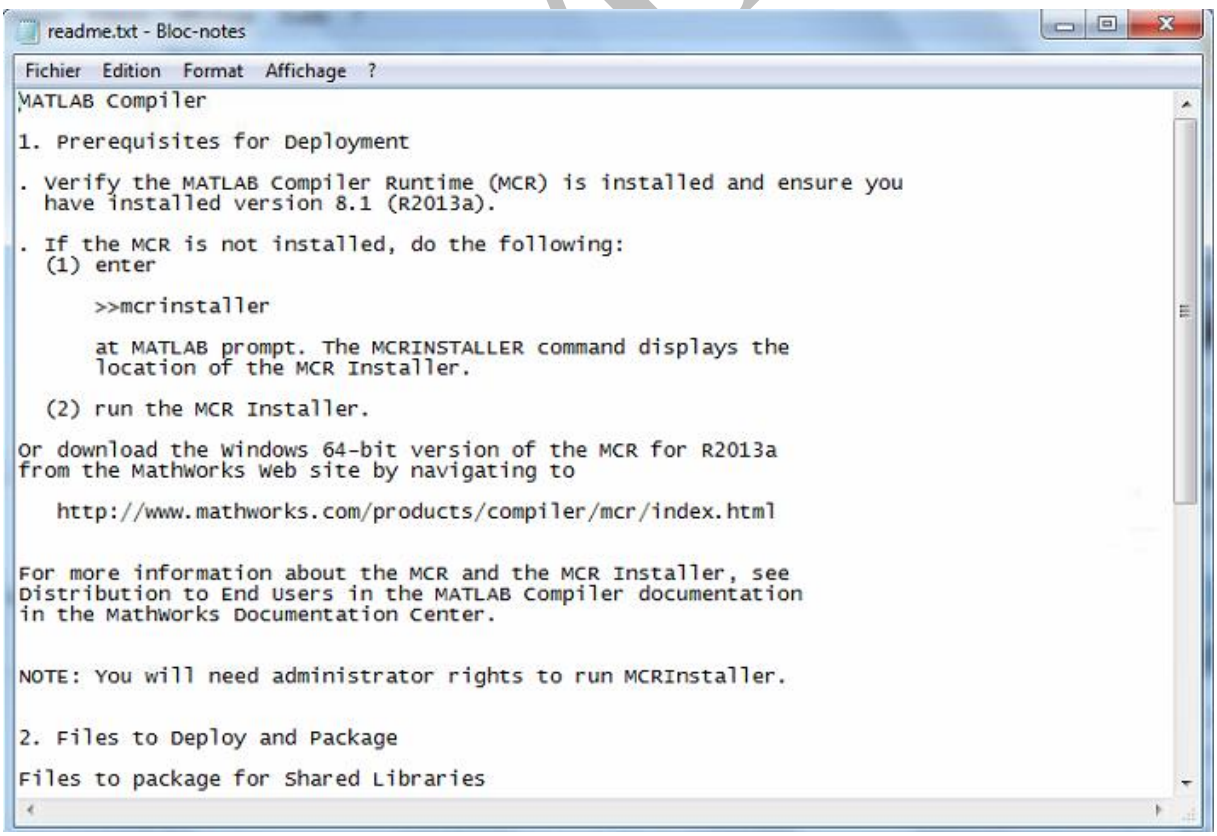

C Shared Library
MATLACProject.prj
Build
Package
Package (1 KB)
MATLACProject.dll
MATLACProject.h
MATLACProject.lib
readme.txt
Add files/folders
Add MCR

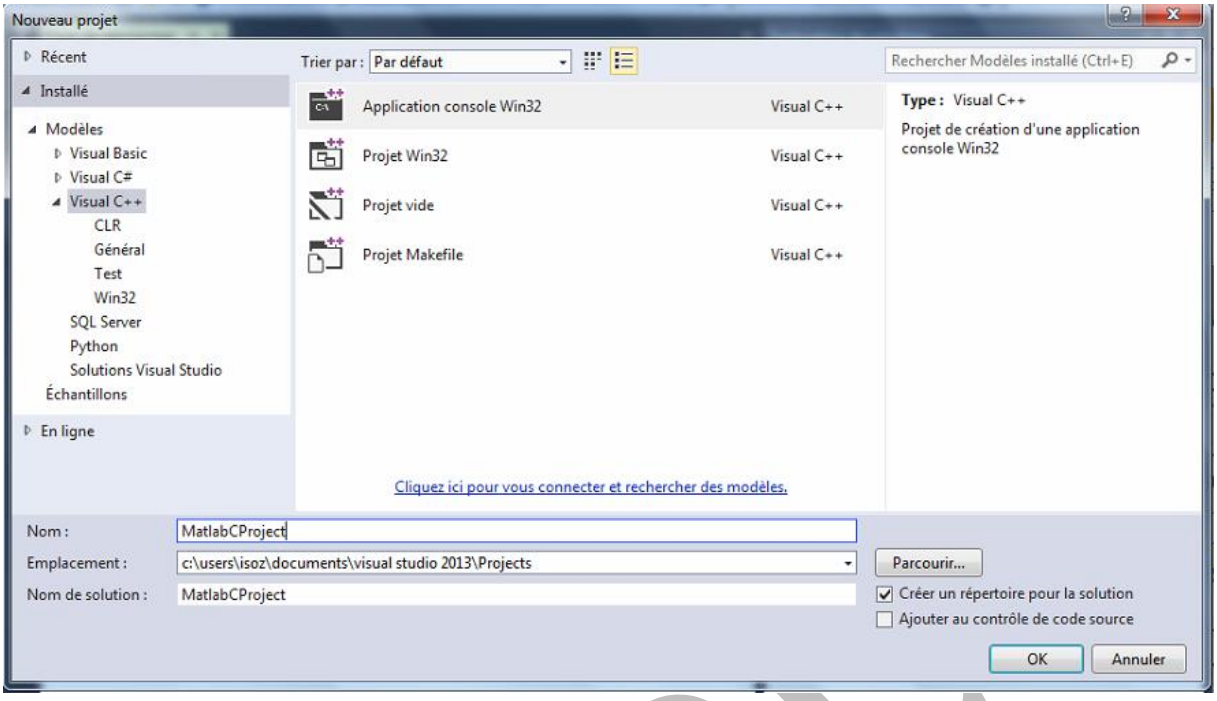
SAMPLE


Build
Build finished.
Hide Details
Close
copy
'C:\tmp\MATLACProject\src\MATLACProject.dll'
'C:\tmp\MATLACProject\distrib\MATLACProject.
copy 'C:\tmp\MATLACProject\src\MATLACProject
'C:\tmp\MATLACProject\distrib\MATLACProject.
copy 'C:\tmp\MATLACProject\src\readme.txt'
'C:\tmp\MATLACProject\distrib\readme.txt'
Save to log file

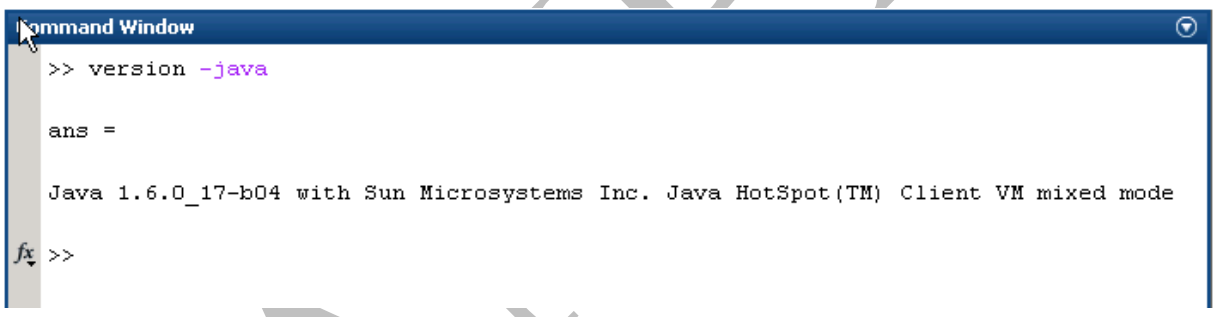

Ordinateur ▸ OS (C:) ▸ tmp ▸ MATLACProject ▸ distrib
Rechercher dans : distrib
Fichier Edition Affichage Outils ?
Organiser Inclure dans la bibliothèque Partager avec Graver Nouveau dossier
dell
Drivers
IDE
Intel
PerfLogs
Program Files
Program Files (x86)
Temp
tmp
MATLACProject
distrib
src
Users
Windows
Réseau
Nom
Modifié le
Type
MATLACProject.dll 25.10.2014 21:42 Extension de l'app...
MATLACProject.h 25.10.2014 21:42 C/C++ Header
MATLACProject.lib 25.10.2014 21:42 Source Browser D...
readme.txt 25.10.2014 21:42 Document texte
4 élément(s)


Ordinateur ▸ OS (C:) ▸ tmp ▸ MATLACProject ▸ src
Rechercher dans : src
Fichier Edition Affichage Outils ?
Organiser Inclure dans la bibliothèque Partager avec Graver Nouveau dossier
dell
Drivers
IDE
Intel
PerfLogs
Program Files
Program Files (x86)
Temp
tmp
MATLACProject
distrib
src
Users
Windows
Réseau
Nom
Modifié le
Type
MATLACProject.c 25.10.2014 21:42 C Source
MATLACProject.dll 25.10.2014 21:42 Extension de l'app...
MATLACProject.exp 25.10.2014 21:42 Source Browser D...
MATLACProject.exports 25.10.2014 21:42 Fichier EXPORTS
MATLACProject.h 25.10.2014 21:42 C/C++ Header
MATLACProject.lib 25.10.2014 21:42 Source Browser D...
mccExcludedFiles.log 25.10.2014 21:42 Document texte
readme.txt 25.10.2014 21:42 Document texte
8 élément(s)


readme.txt - Bloc-notes
Fichier Edition Format Affichage ?
MATLAB Compiler
1. Prerequisites for Deployment
. Verify the MATLAB Compiler Runtime (MCR) is installed and ensure you
have installed version 8.1 (R2013a).
. If the MCR is not installed, do the following:
(1) enter
>>mcrinstaller
at MATLAB prompt. The MCRINSTALLER command displays the
location of the MCR Installer.
(2) run the MCR Installer.
Or download the Windows 64-bit version of the MCR for R2013a
from the MathWorks Web site by navigating to
http://www.mathworks.com/products/compiler/mcr/index.html
For more information about the MCR and the MCR Installer, see
Distribution to End Users in the MATLAB Compiler documentation
in the MathWorks Documentation Center.
NOTE: You will need administrator rights to run MCRInstaller.
2. Files to Deploy and Package
Files to package for Shared Libraries

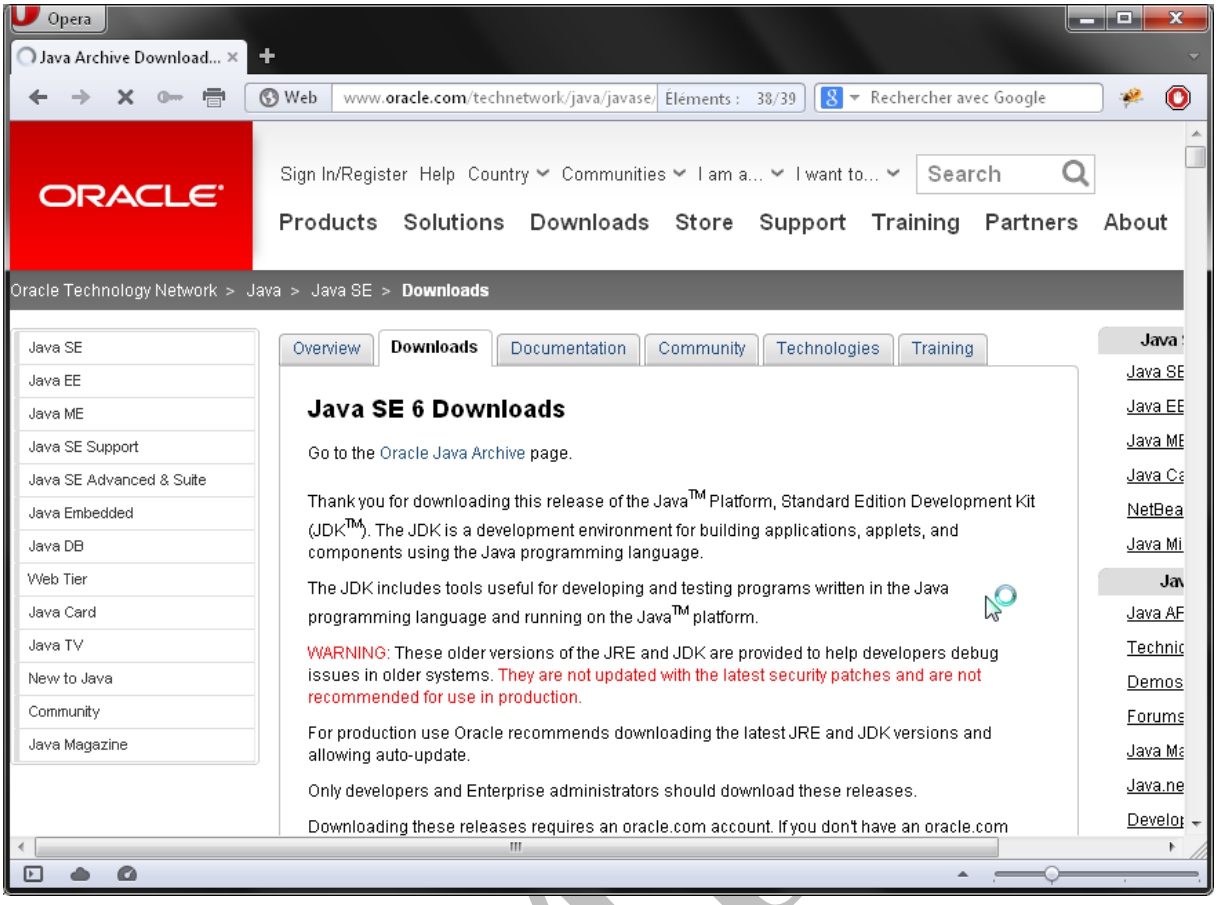


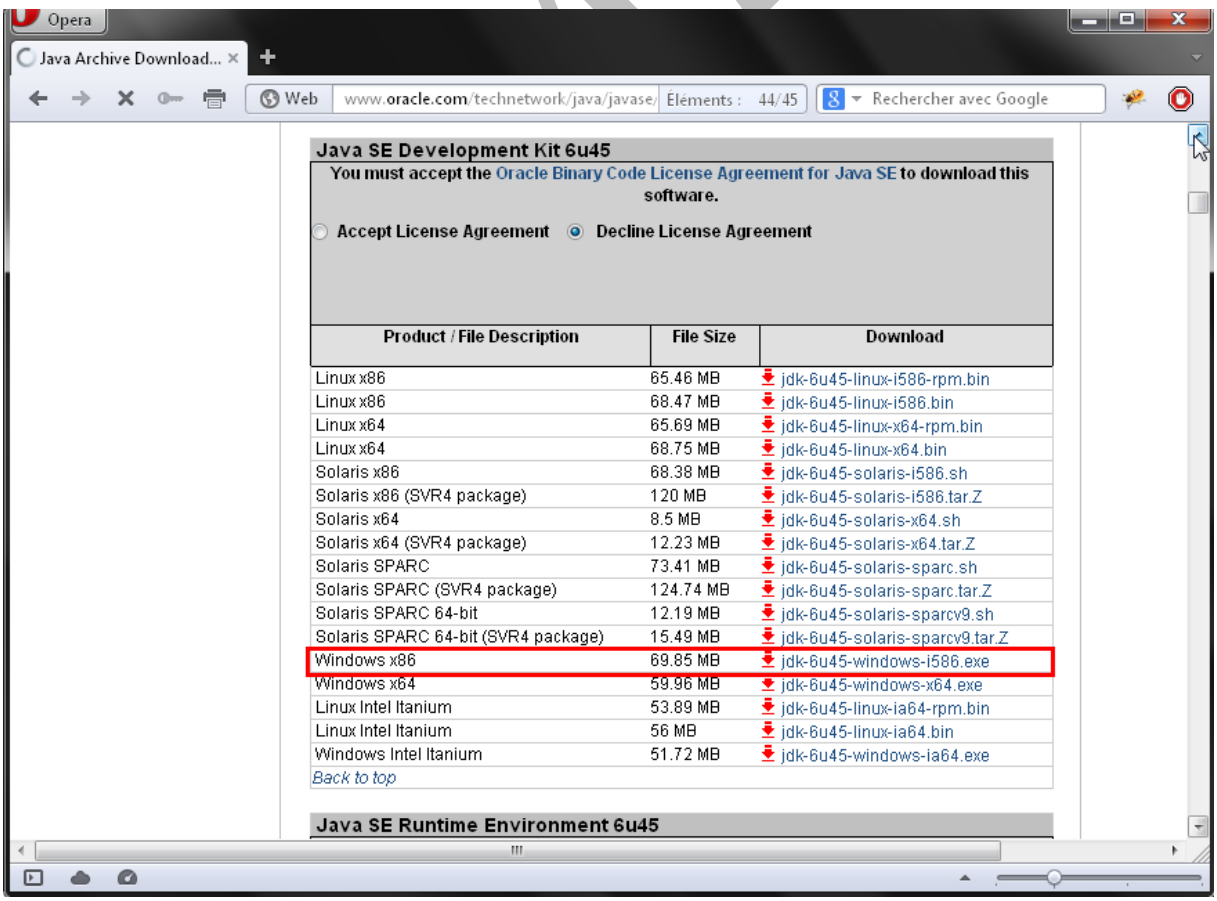
Compile JAR and call from JAVA

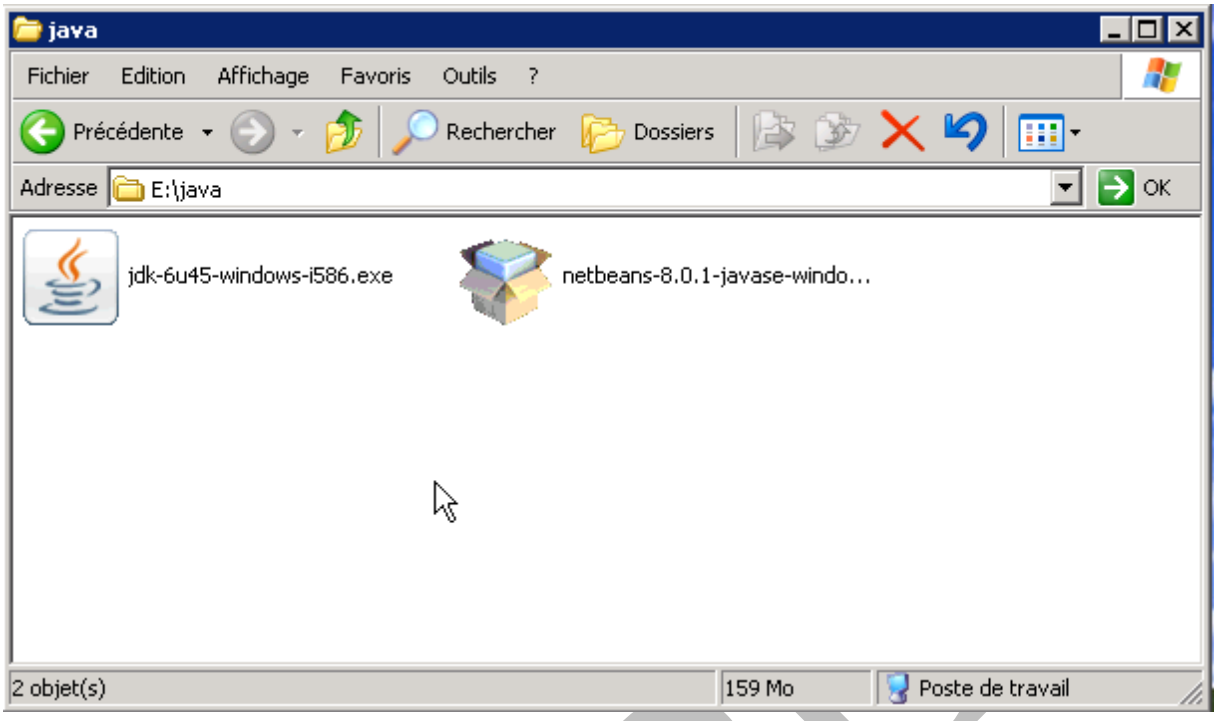


http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html#jdk-6u45-oth-JPR

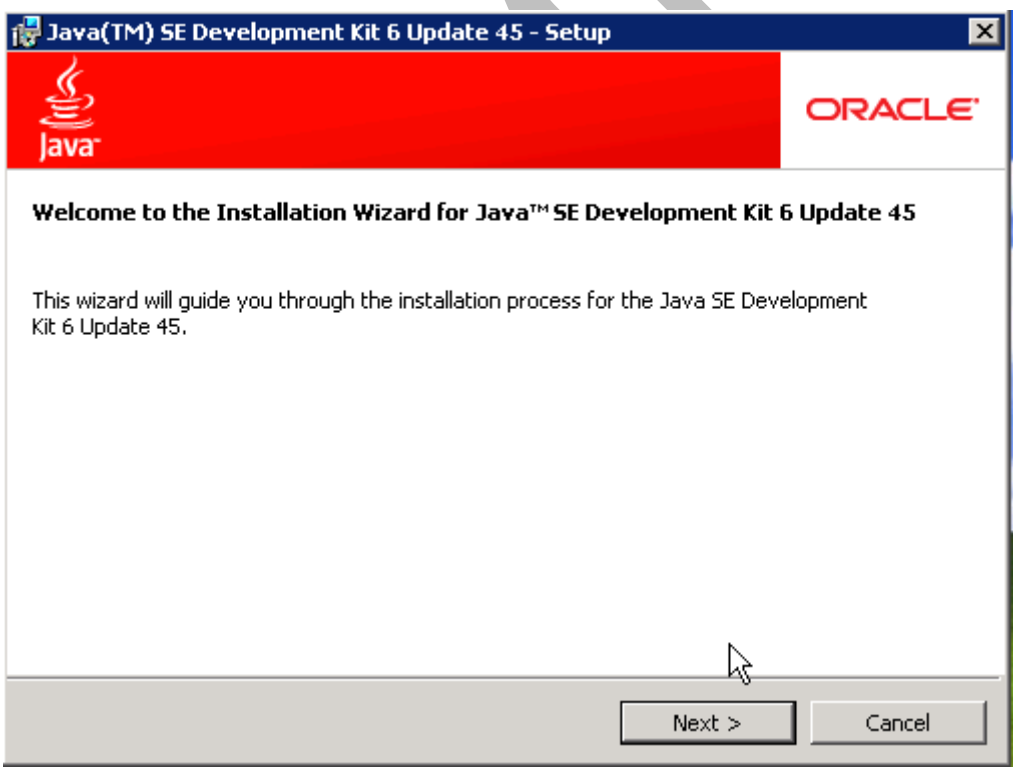
SAMPLE

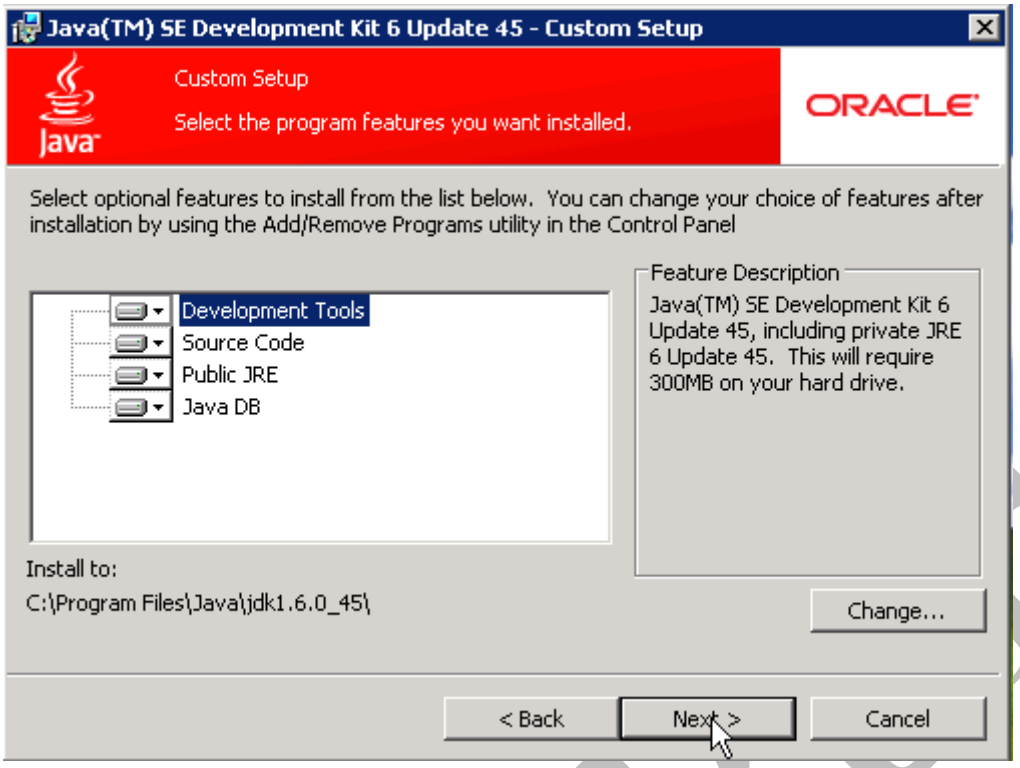

Opera
Java Archive Download...
www.oracle.com/technetwork/java/javase/
Sign In/Register Help Country Communities I am a... I want to... Search
Products Solutions Downloads Store Support Training Partners About
Oracle Technology Network > Java > Java SE > Downloads
Java SE
Java EE
Java ME
Java SE Support
Java SE Advanced & Suite
Java Embedded
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine
Overview Downloads Documentation Community Technologies Training
Java SE 6 Downloads
Go to the Oracle Java Archive page.
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.
The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java™ platform.
WARNING: These older versions of the JRE and JDK are provided to help developers debug issues in older systems. They are not updated with the latest security patches and are not recommended for use in production.
For production use Oracle recommends downloading the latest JRE and JDK versions and allowing auto-update.
Only developers and Enterprise administrators should download these releases.
Downloading these releases requires an oracle.com account. If you don't have an oracle.com

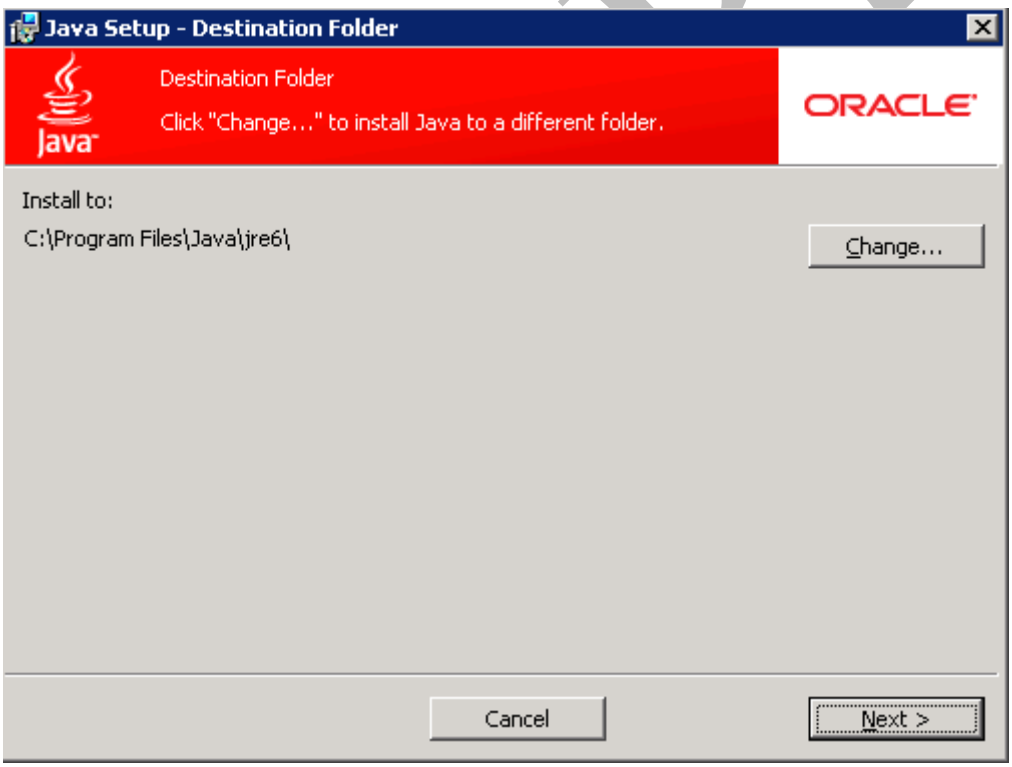

Opera
Java Archive Download...
www.oracle.com/technetwork/java/javase/
Java SE Development Kit 6u45
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Accept License Agreement Decline License Agreement
Product / File Description | File Size | Download
Linux x86 | 65.46 MB | jdk-6u45-linux-i586-rpm.bin
Linux x86 | 68.47 MB | jdk-6u45-linux-i586.bin
Linux x64 | 65.69 MB | jdk-6u45-linux-x64-rpm.bin
Linux x64 | 68.75 MB | jdk-6u45-linux-x64.bin
Solaris x86 | 68.38 MB | jdk-6u45-solaris-i586.sh
Solaris x86 (SVR4 package) | 120 MB | jdk-6u45-solaris-i586.tar.Z
Solaris x64 | 8.5 MB | jdk-6u45-solaris-x64.sh
Solaris x64 (SVR4 package) | 12.23 MB | jdk-6u45-solaris-x64.tar.Z
Solaris SPARC | 73.41 MB | jdk-6u45-solaris-sparc.sh
Solaris SPARC (SVR4 package) | 124.74 MB | jdk-6u45-solaris-sparc.tar.Z
Solaris SPARC 64-bit | 12.19 MB | jdk-6u45-solaris-sparcv9.sh
Solaris SPARC 64-bit (SVR4 package) | 15.49 MB | jdk-6u45-solaris-sparcv9.tar.Z
Windows x86 | 69.85 MB | jdk-6u45-windows-i586.exe
Windows x64 | 59.96 MB | jdk-6u45-windows-x64.exe
Linux Intel Itanium | 53.89 MB | jdk-6u45-linux-ia64-rpm.bin
Linux Intel Itanium | 56 MB | jdk-6u45-linux-ia64.bin
Windows Intel Itanium | 51.72 MB | jdk-6u45-windows-ia64.exe
Back to top
Java SE Runtime Environment 6u45

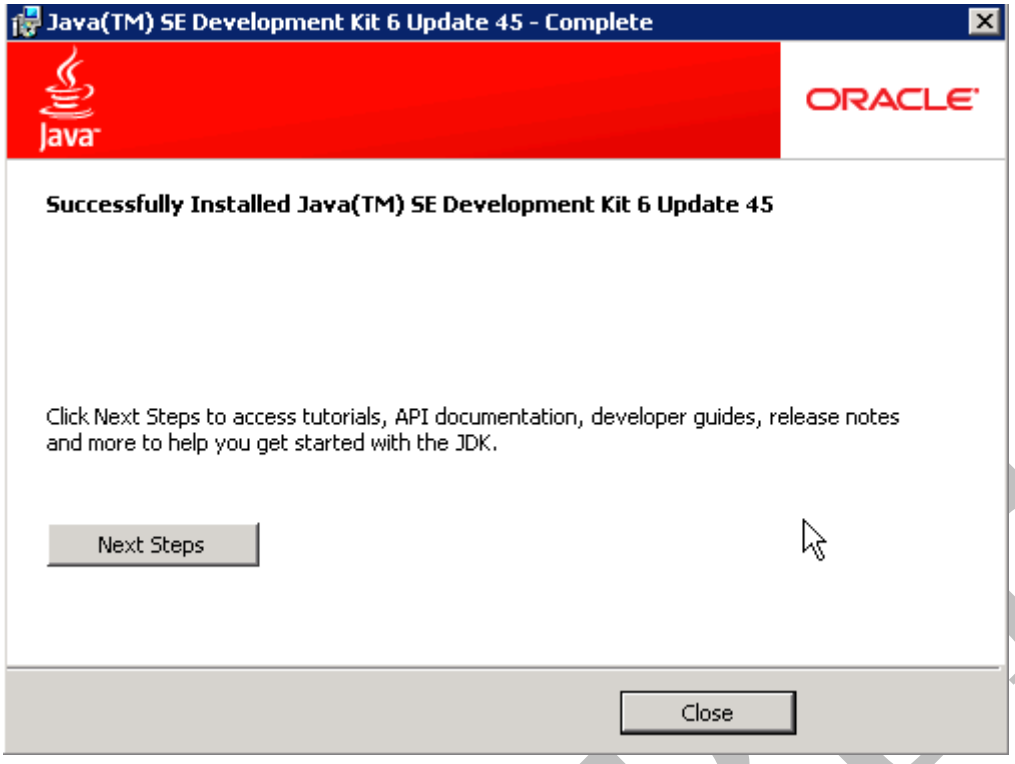


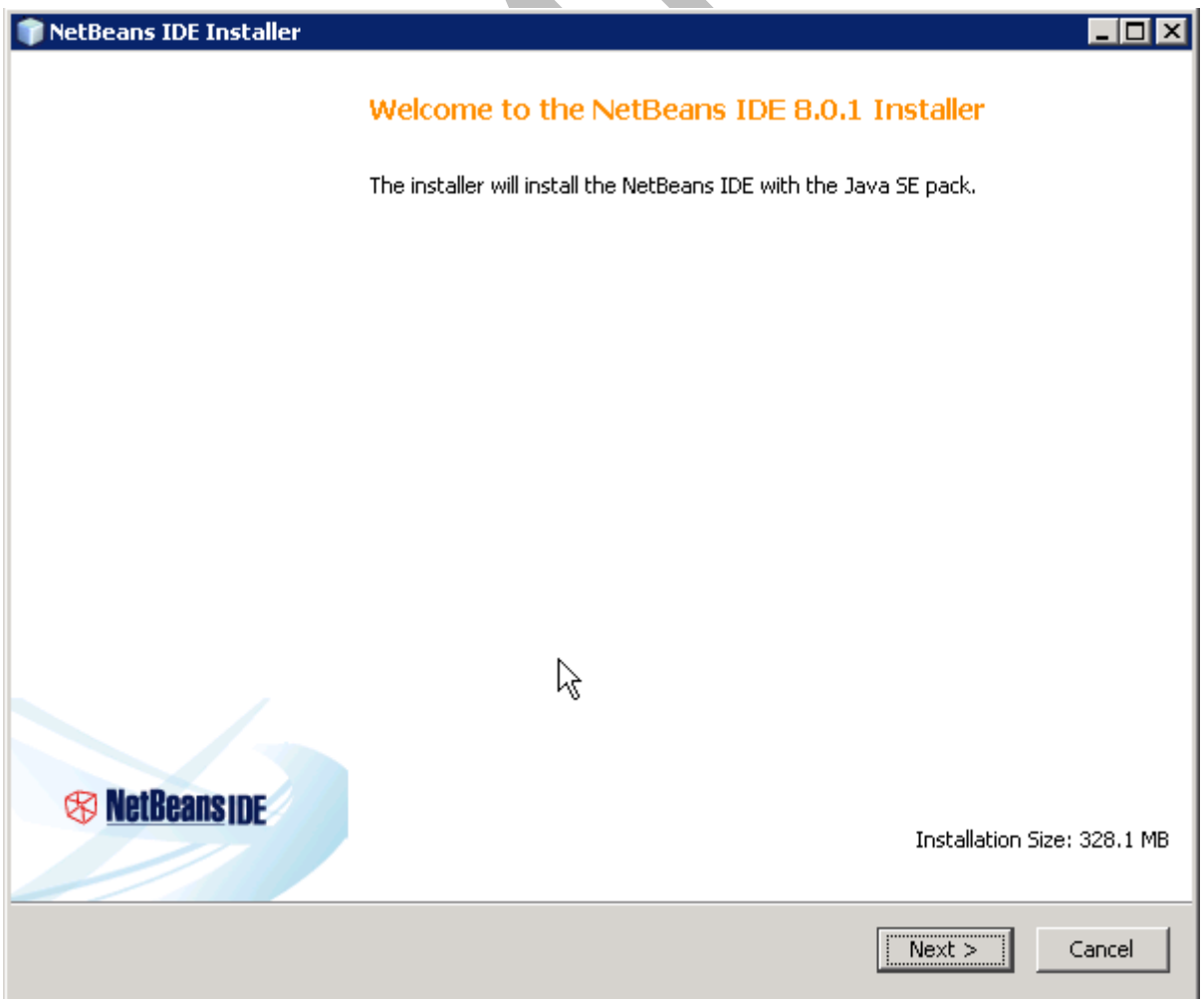
For MATLAB 2013a install JDK 6.45 but for NetBeans install at least the JDK 7.0 (it's not a problem to have more than a version install on the same computer):

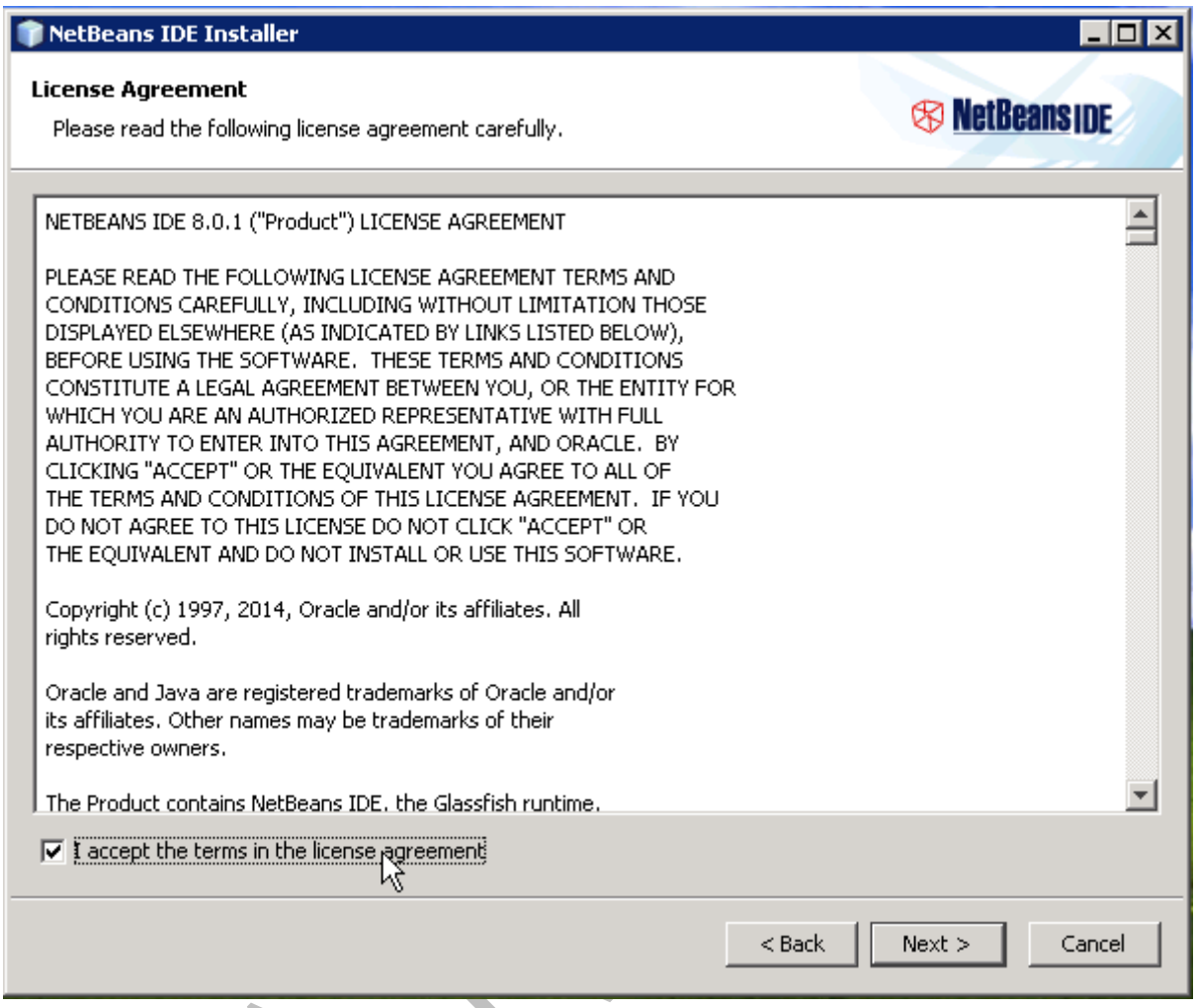



Java(TM) SE Development Kit 6 Update 45 - Custom Setup
Custom Setup
Select the program features you want installed.
ORACLE
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Source Code
Public JRE
Java DB
Feature Description
Java(TM) SE Development Kit 6 Update 45, including private JRE 6 Update 45. This will require 300MB on your hard drive.
Install to:
C:\Program Files\Java\jdk1.6.0_45\
Change...
< Back
Next >
Cancel

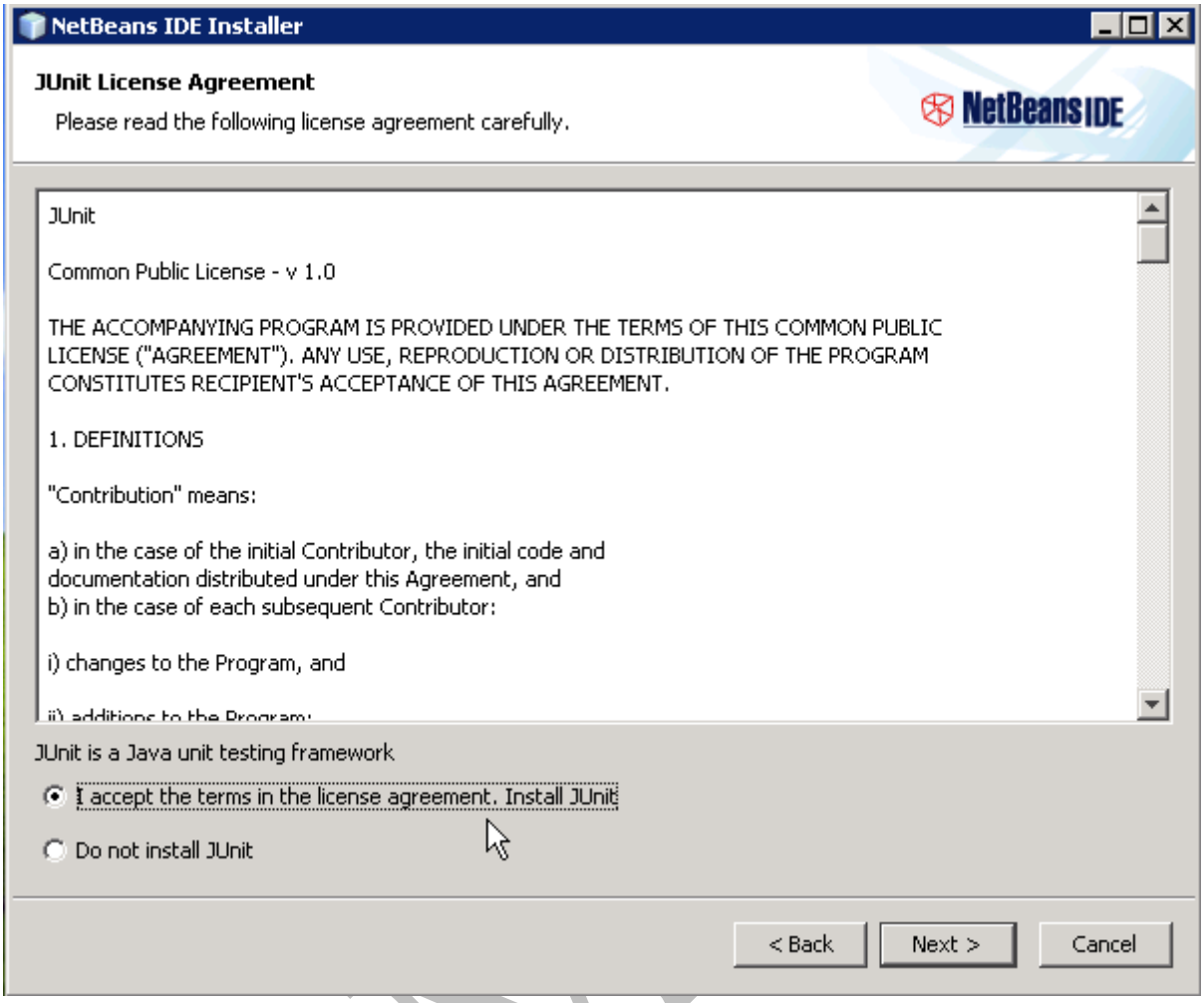

Java Setup - Destination Folder
Destination Folder
Click "Change..." to install Java to a different folder.
ORACLE
Install to:
C:\Program Files\Java\jre6\
Change...
Cancel
Next >


Java(TM) SE Development Kit 6 Update 45 - Complete
Java
ORACLE
Successfully Installed Java(TM) SE Development Kit 6 Update 45
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close

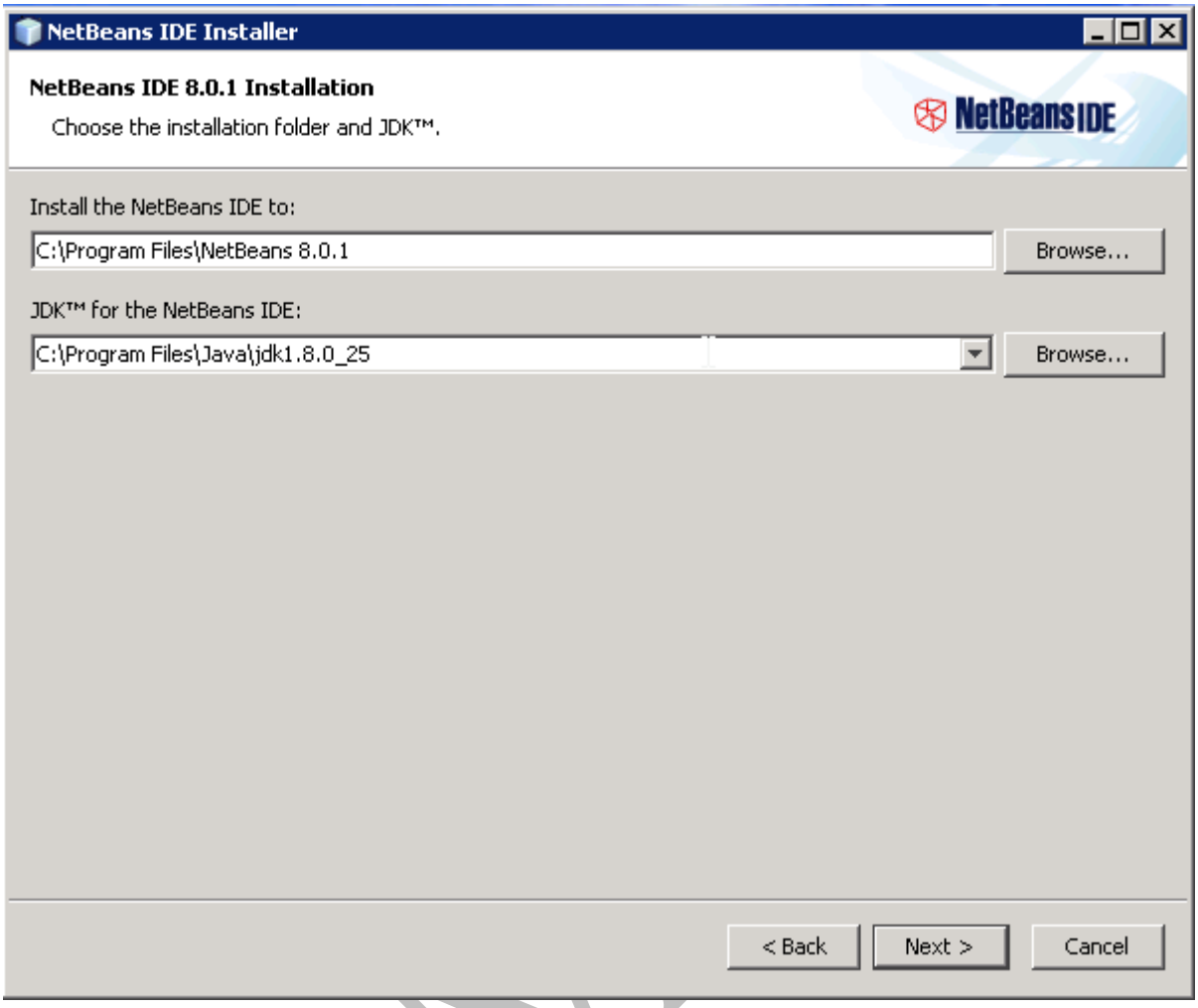

NetBeans IDE Installer
Welcome to the NetBeans IDE 8.0.1 Installer
The installer will install the NetBeans IDE with the Java SE pack.
NetBeans IDE
Installation Size: 328.1 MB
Next >
Cancel


NetBeans IDE Installer
License Agreement
Please read the following license agreement carefully.
NetBeans IDE
NETBEANS IDE 8.0.1 ("Product") LICENSE AGREEMENT
PLEASE READ THE FOLLOWING LICENSE AGREEMENT TERMS AND CONDITIONS CAREFULLY, INCLUDING WITHOUT LIMITATION THOSE DISPLAYED ELSEWHERE (AS INDICATED BY LINKS LISTED BELOW), BEFORE USING THE SOFTWARE. THESE TERMS AND CONDITIONS CONSTITUTE A LEGAL AGREEMENT BETWEEN YOU, OR THE ENTITY FOR WHICH YOU ARE AN AUTHORIZED REPRESENTATIVE WITH FULL AUTHORITY TO ENTER INTO THIS AGREEMENT, AND ORACLE. BY CLICKING "ACCEPT" OR THE EQUIVALENT YOU AGREE TO ALL OF THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THIS LICENSE DO NOT CLICK "ACCEPT" OR THE EQUIVALENT AND DO NOT INSTALL OR USE THIS SOFTWARE.
Copyright (c) 1997, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
The Product contains NetBeans IDE, the Glassfish runtime,
I accept the terms in the license agreement
< Back
Next >
Cancel

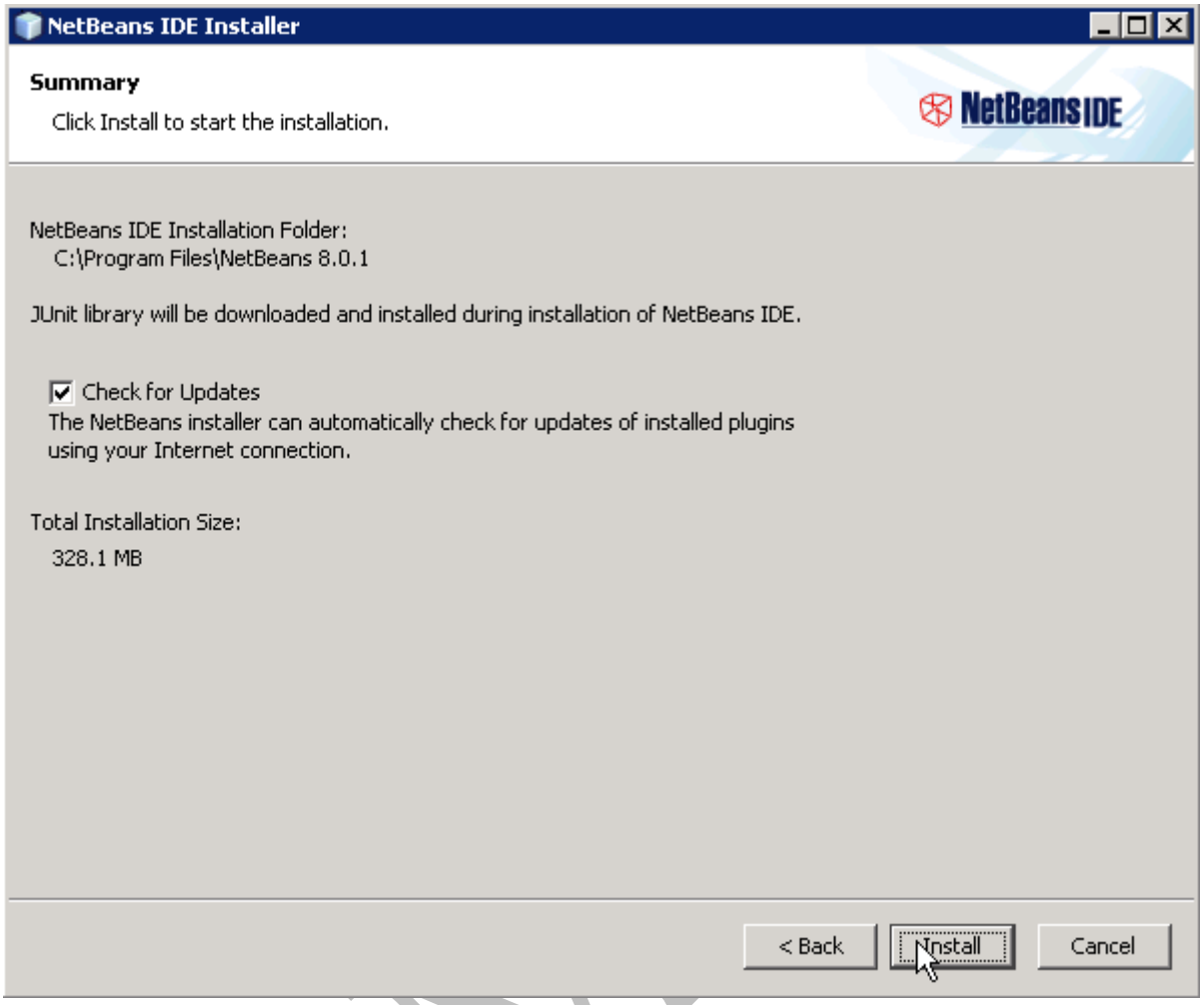
SAMPLE


NetBeans IDE Installer
JUnit License Agreement
Please read the following license agreement carefully.
NetBeans IDE
JUnit
Common Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.
1. DEFINITIONS
"Contribution" means:
a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
JUnit is a Java unit testing framework
I accept the terms in the license agreement. Install JUnit
Do not install JUnit
< Back
Next >
Cancel

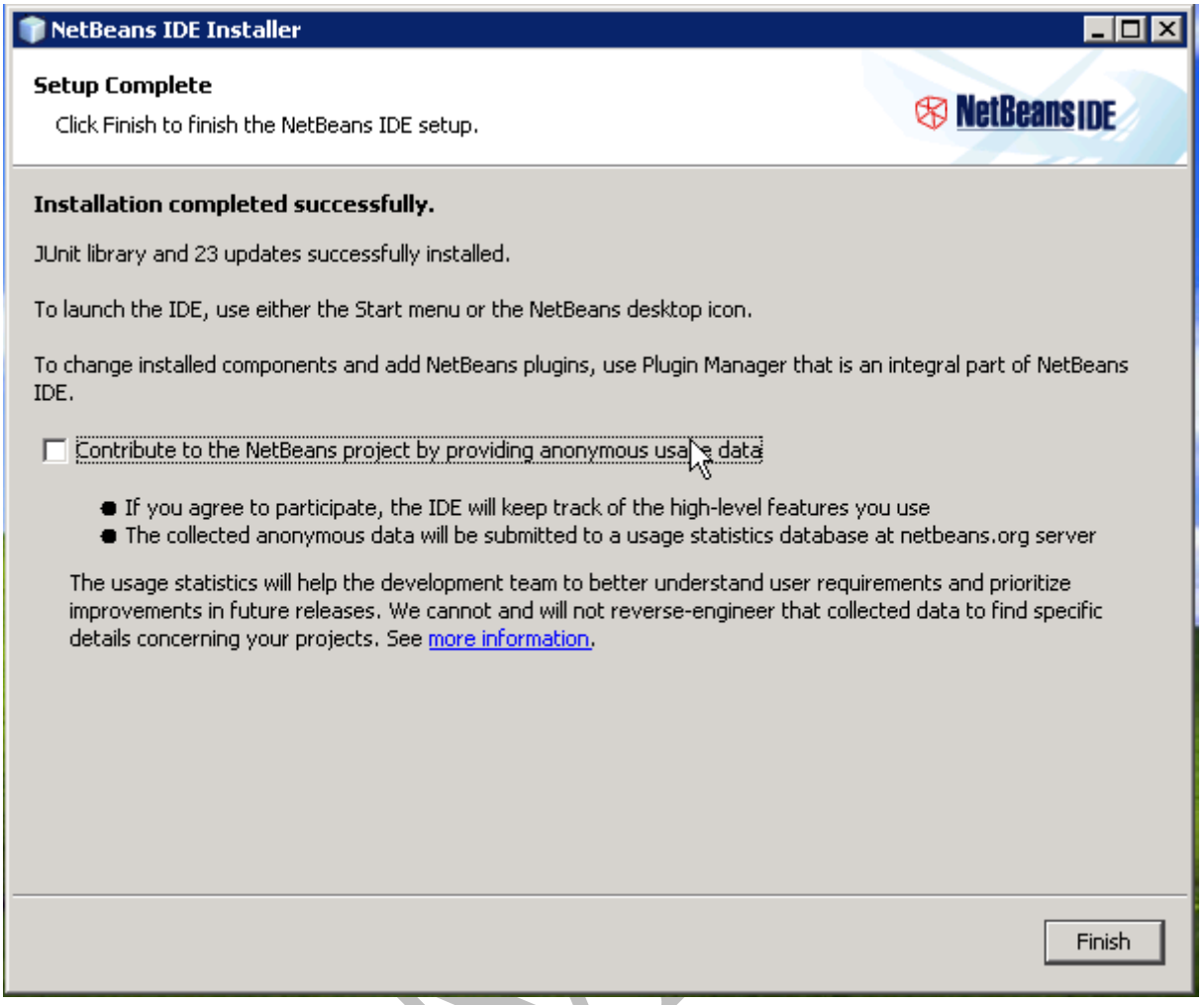
SAMPLE


NetBeans IDE Installer
NetBeans IDE 8.0.1 Installation
Choose the installation folder and JDK™.
NetBeans IDE
Install the NetBeans IDE to:
C:\Program Files\NetBeans 8.0.1
Browse...
JDK™ for the NetBeans IDE:
C:\Program Files\Java\jdk1.8.0_25
Browse...
< Back
Next >
Cancel

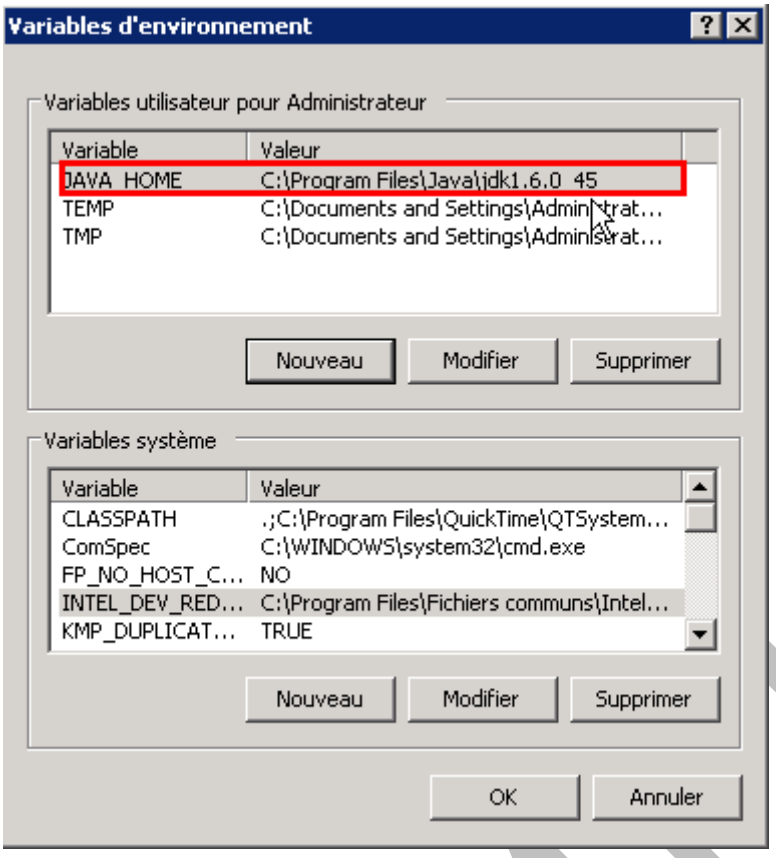
SAM

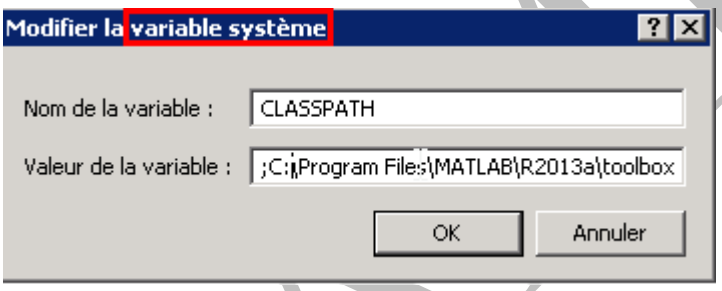

NetBeans IDE Installer
Summary
Click Install to start the installation.
NetBeans IDE
NetBeans IDE Installation Folder:
C:\Program Files\NetBeans 8.0.1
JUnit library will be downloaded and installed during installation of NetBeans IDE.
Check for Updates
The NetBeans installer can automatically check for updates of installed plugins using your Internet connection.
Total Installation Size:
328.1 MB
< Back
Install
Cancel

SAMPLE

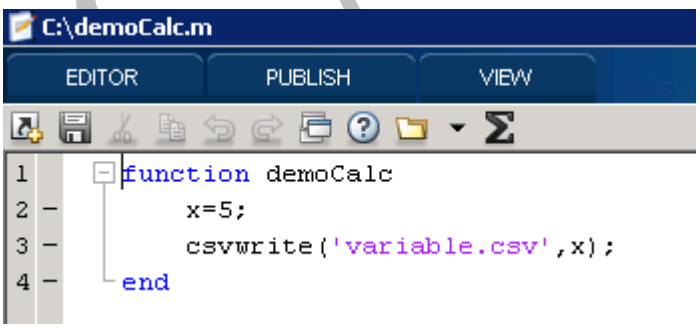

NetBeans IDE Installer
Setup Complete
Click Finish to finish the NetBeans IDE setup.
NetBeans IDE
Installation completed successfully.
JUnit library and 23 updates successfully installed.
To launch the IDE, use either the Start menu or the NetBeans desktop icon.
To change installed components and add NetBeans plugins, use Plugin Manager that is an integral part of NetBeans IDE.
Contribute to the NetBeans project by providing anonymous usage data
If you agree to participate, the IDE will keep track of the high-level features you use
The collected anonymous data will be submitted to a usage statistics database at netbeans.org server
The usage statistics will help the development team to better understand user requirements and prioritize improvements in future releases. We cannot and will not reverse-engineer that collected data to find specific details concerning your projects. See more information.
Finish

SAM

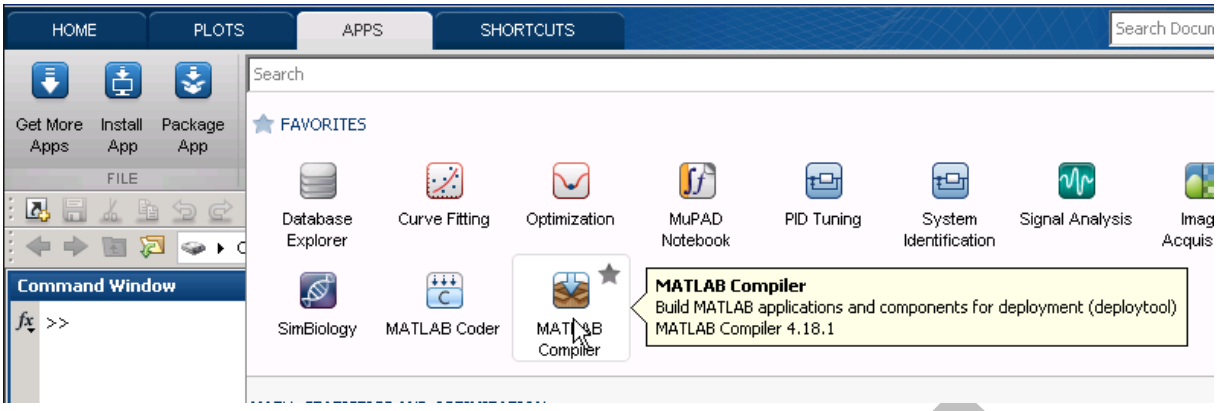


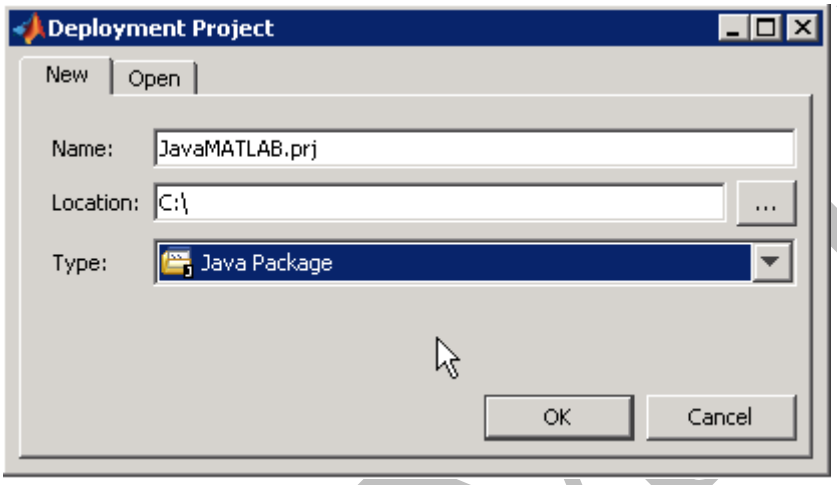


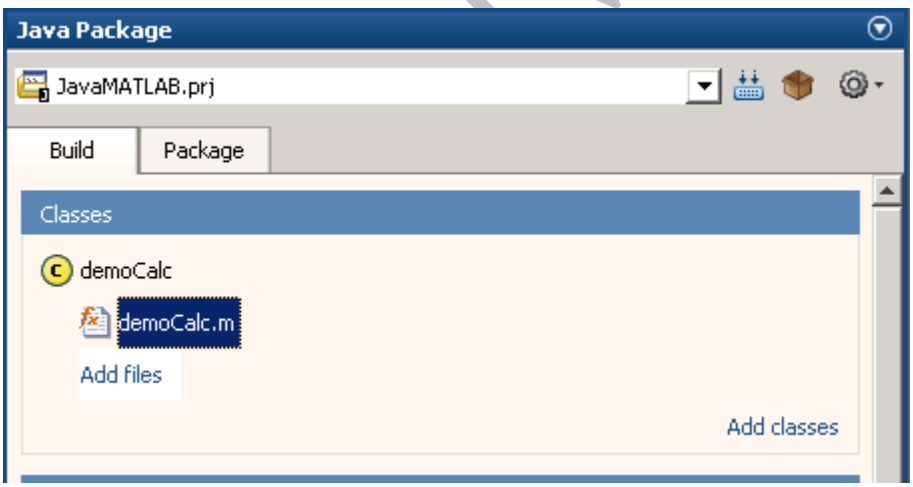
Once this is done don't forget to **restart you computer** (sometimes it helps to avoid issues during compilation).

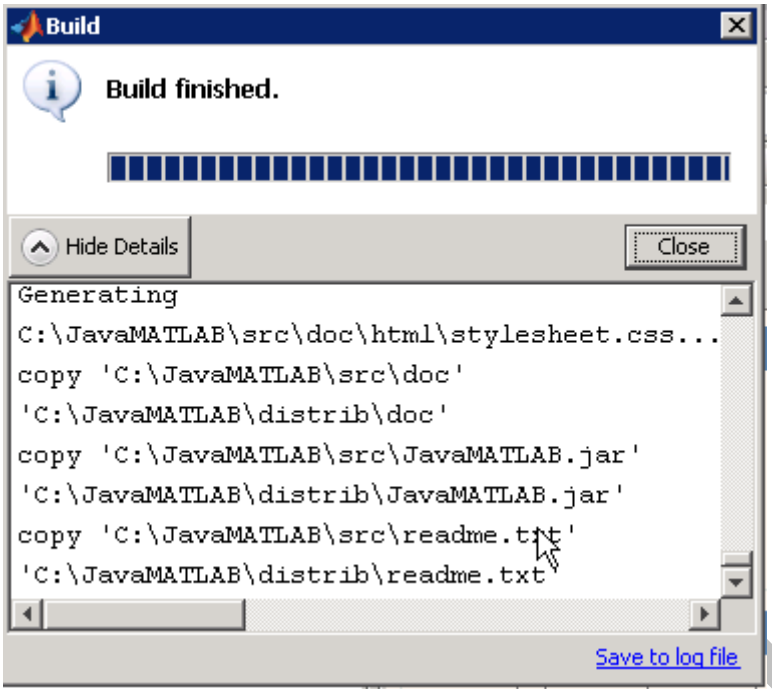


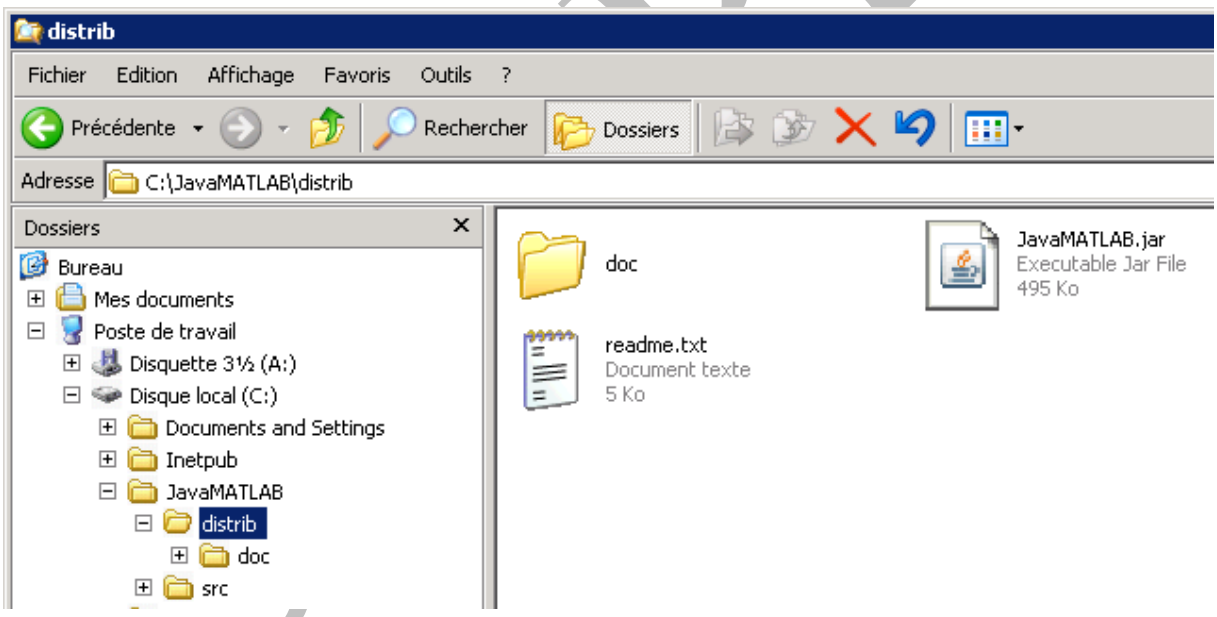
SAMPLE

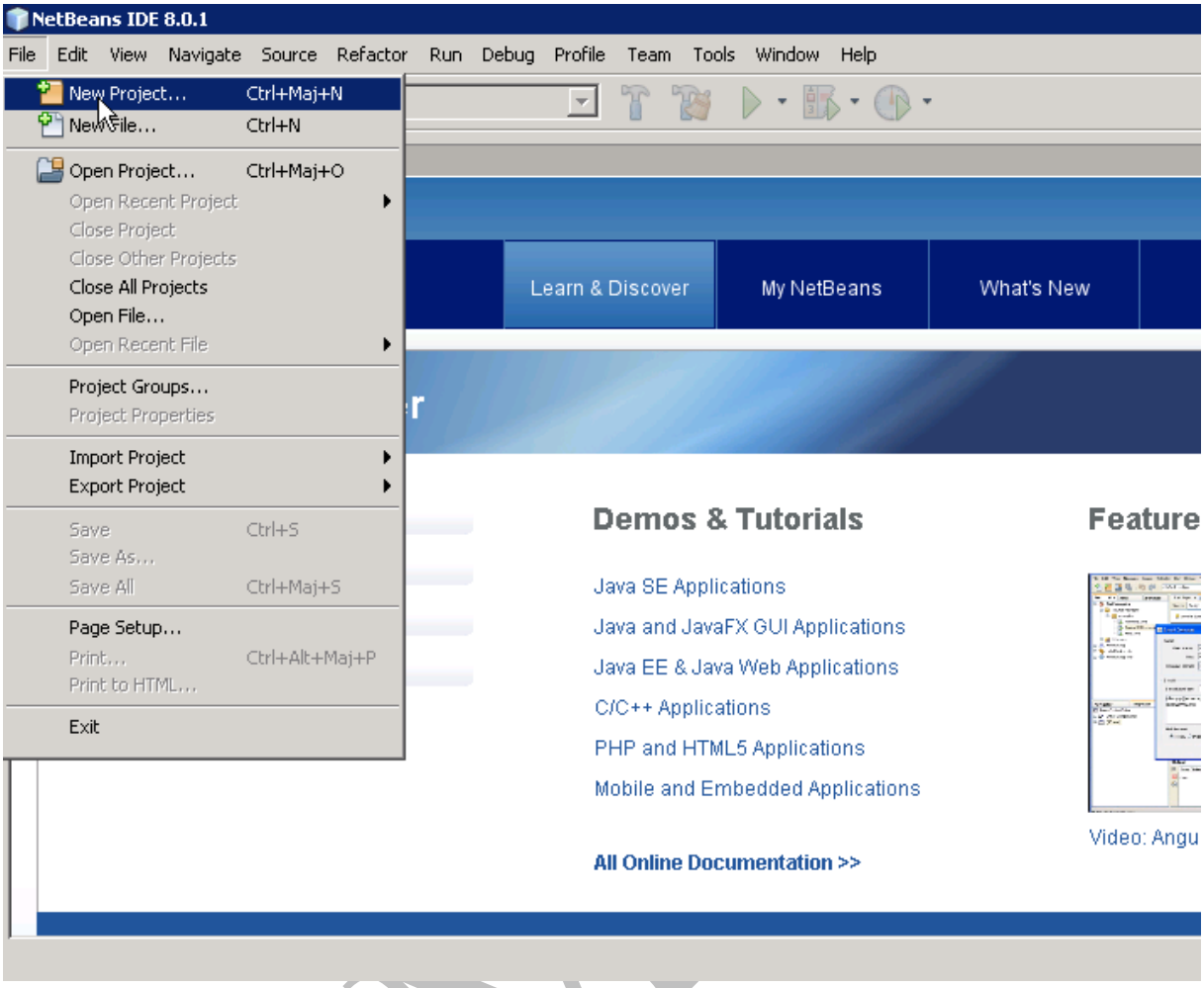

HOME
PLOTS
APPS
SHORTCUTS
Search Docun
Get More Apps
Install App
Package App
FILE
Search
FAVORITES
Database Explorer
Curve Fitting
Optimization
MuPAD Notebook
PID Tuning
System Identification
Signal Analysis
Imag Acquis
Command Window
>>
SimBiology
MATLAB Coder
MATLAB Compiler
MATLAB Compiler
Build MATLAB applications and components for deployment (deploytool)
MATLAB Compiler 4.18.1

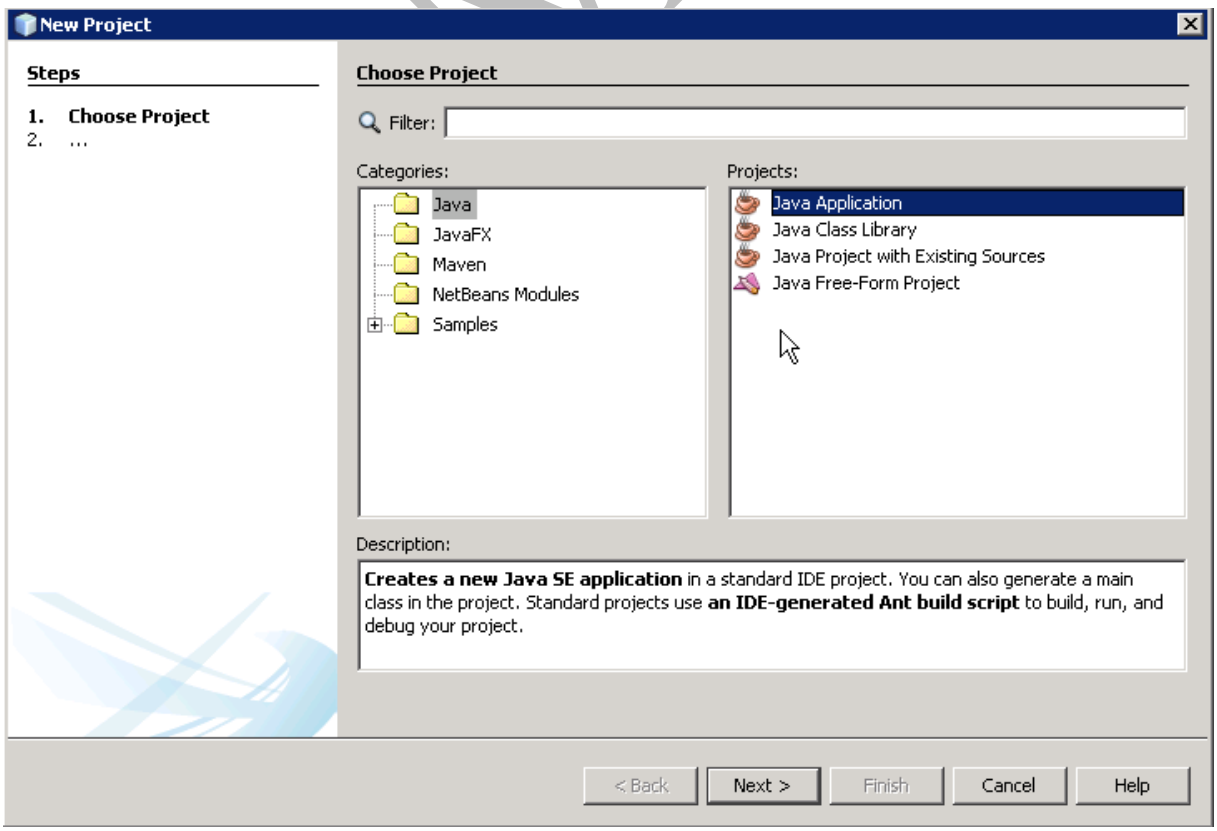

Deployment Project
New
Open
Name:
JavaMATLAB.prj
Location:
C:\
...
Type:
Java Package
OK
Cancel

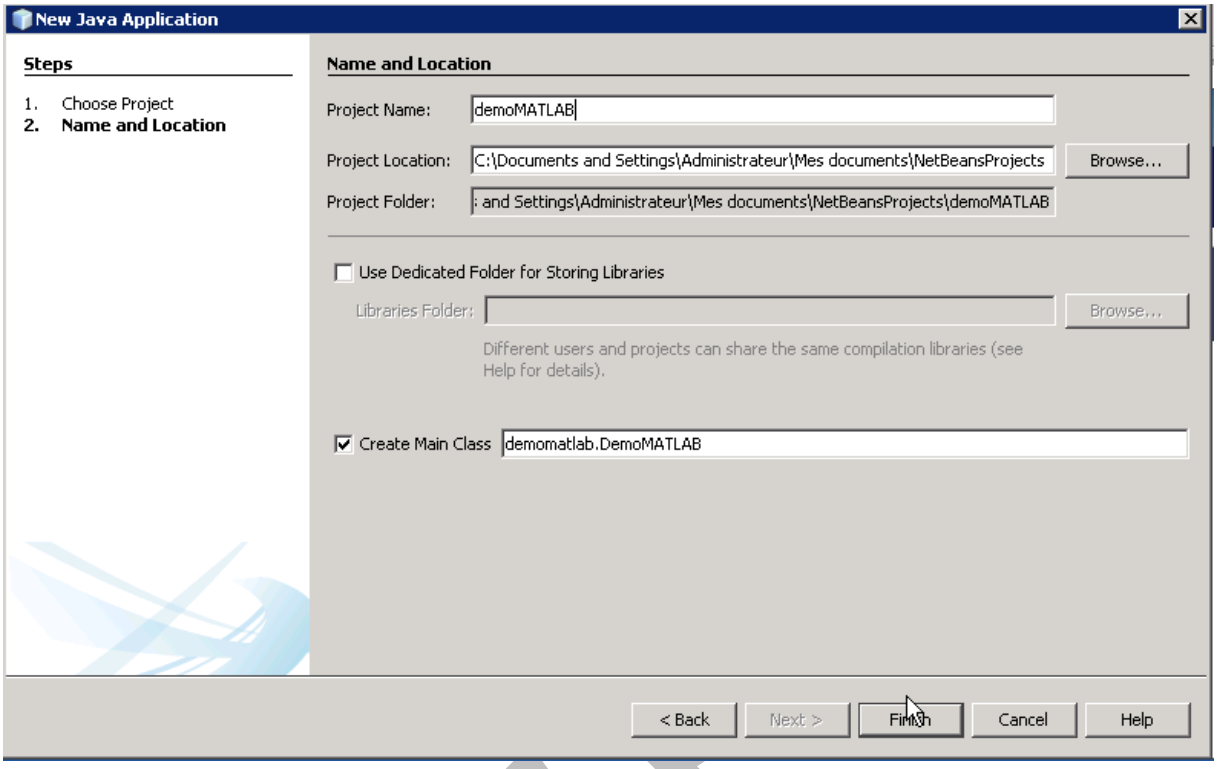

Java Package
JavaMATLAB.prj
Build
Package
Classes
demoCalc
demoCalc.m
Add files
Add classes

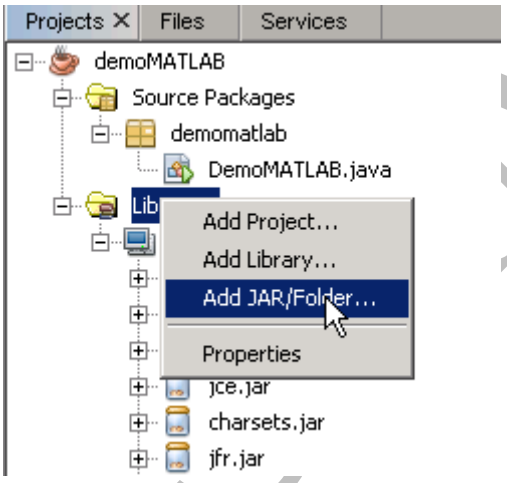

Build
Build finished.
Hide Details
Close
Generating
C:\JavaMATLAB\src\doc\html\stylesheet.css...
copy 'C:\JavaMATLAB\src\doc'
'C:\JavaMATLAB\distrib\doc'
copy 'C:\JavaMATLAB\src\JavaMATLAB.jar'
'C:\JavaMATLAB\distrib\JavaMATLAB.jar'
copy 'C:\JavaMATLAB\src\readme.txt'
'C:\JavaMATLAB\distrib\readme.txt'
Save to log file

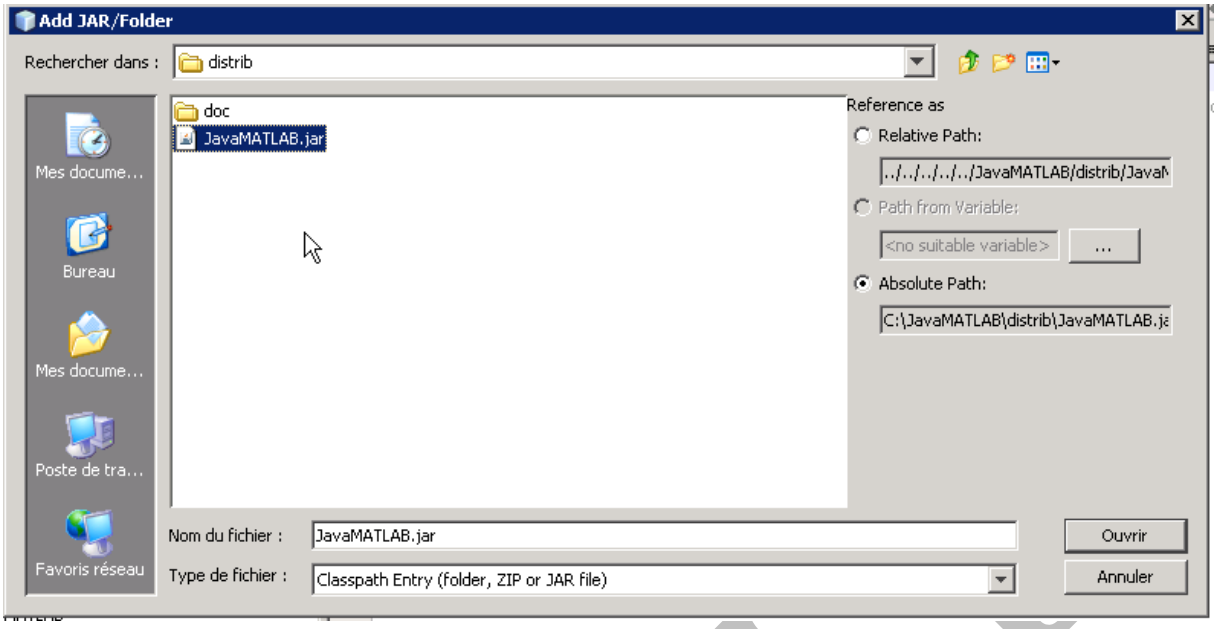

distrib
Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Dossiers
Adresse C:\JavaMATLAB\distrib
Dossiers
Bureau
Mes documents
Poste de travail
Disquette 3½ (A:)
Disque local (C:)
Documents and Settings
Inetpub
JavaMATLAB
distrib
doc
src
doc
JavaMATLAB.jar
Executable Jar File
495 Ko
readme.txt
Document texte
5 Ko


NetBeans IDE 8.0.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
New Project... Ctrl+Maj+N
New File... Ctrl+N
Open Project... Ctrl+Maj+O
Open Recent Project
Close Project
Close Other Projects
Close All Projects
Open File...
Open Recent File
Project Groups...
Project Properties
Import Project
Export Project
Save Ctrl+S
Save As...
Save All Ctrl+Maj+S
Page Setup...
Print... Ctrl+Alt+Maj+P
Print to HTML...
Exit
Learn & Discover
My NetBeans
What's New
Demos & Tutorials
Feature
Java SE Applications
Java and JavaFX GUI Applications
Java EE & Java Web Applications
C/C++ Applications
PHP and HTML5 Applications
Mobile and Embedded Applications
All Online Documentation >>
Video: Angu

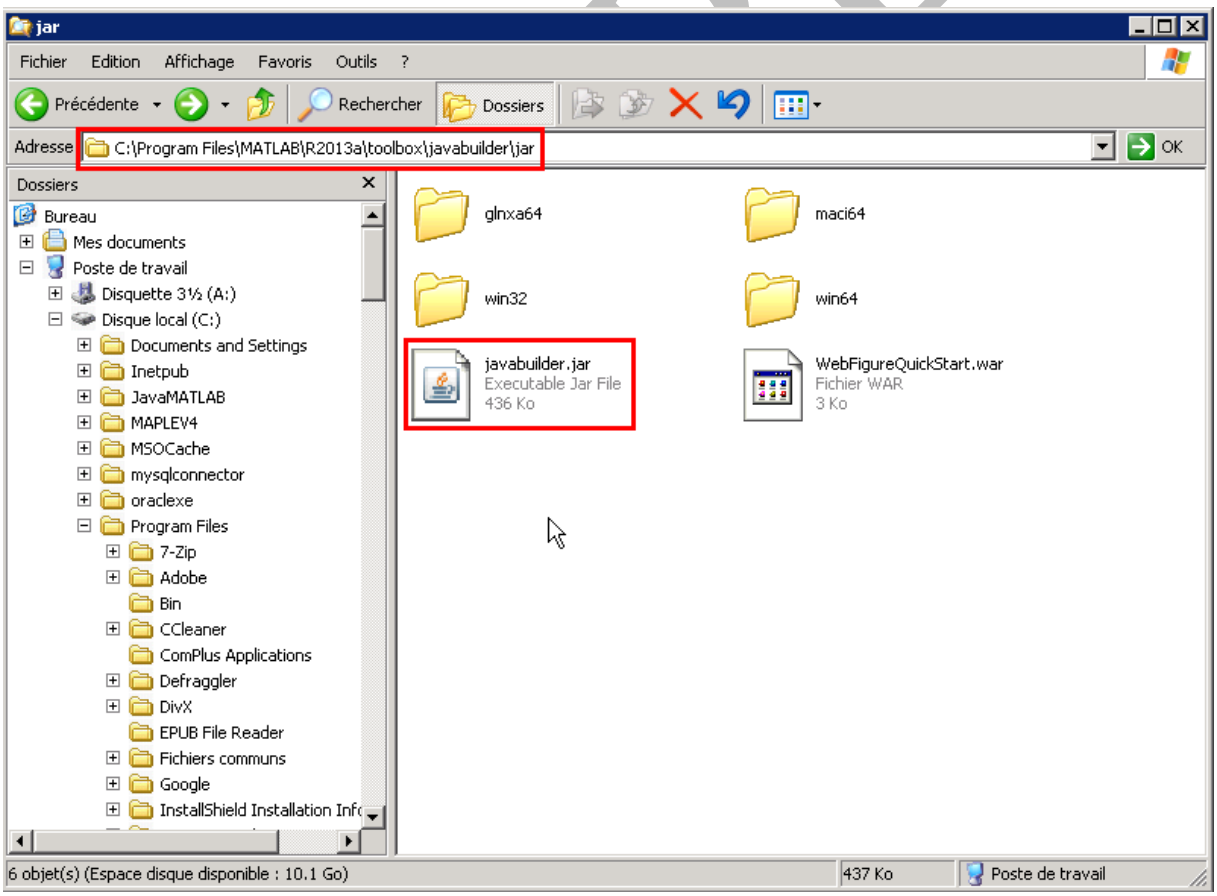

New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java
JavaFX
Maven
NetBeans Modules
Samples
Projects:
Java Application
Java Class Library
Java Project with Existing Sources
Java Free-Form Project
Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use an IDE-generated Ant build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help


New Java Application
Steps
1. Choose Project
2. Name and Location
Name and Location
Project Name: demoMATLAB
Project Location: C:\Documents and Settings\Administrateur\Mes documents\NetBeansProjects
Browse...
Project Folder: : and Settings\Administrateur\Mes documents\NetBeansProjects\demoMATLAB
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
Create Main Class demomatlab.DemoMATLAB
< Back
Next >
Finish
Cancel
Help

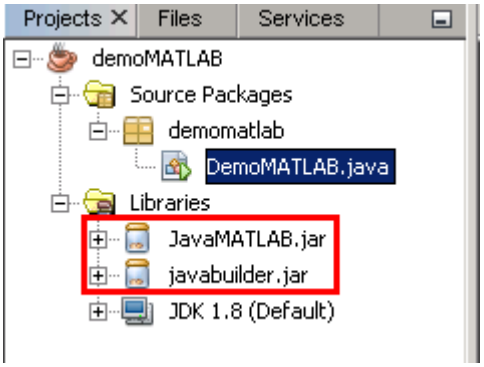

Projects
Files
Services
demoMATLAB
Source Packages
demomatlab
DemoMATLAB.java
Lib
Add Project...
Add Library...
Add JAR/Folder...
Properties
jce.jar
charsets.jar
jfr.jar



To avoid some problems on some computers also add as reference the MATLAB javabulilder.jar for the toolbox folder:



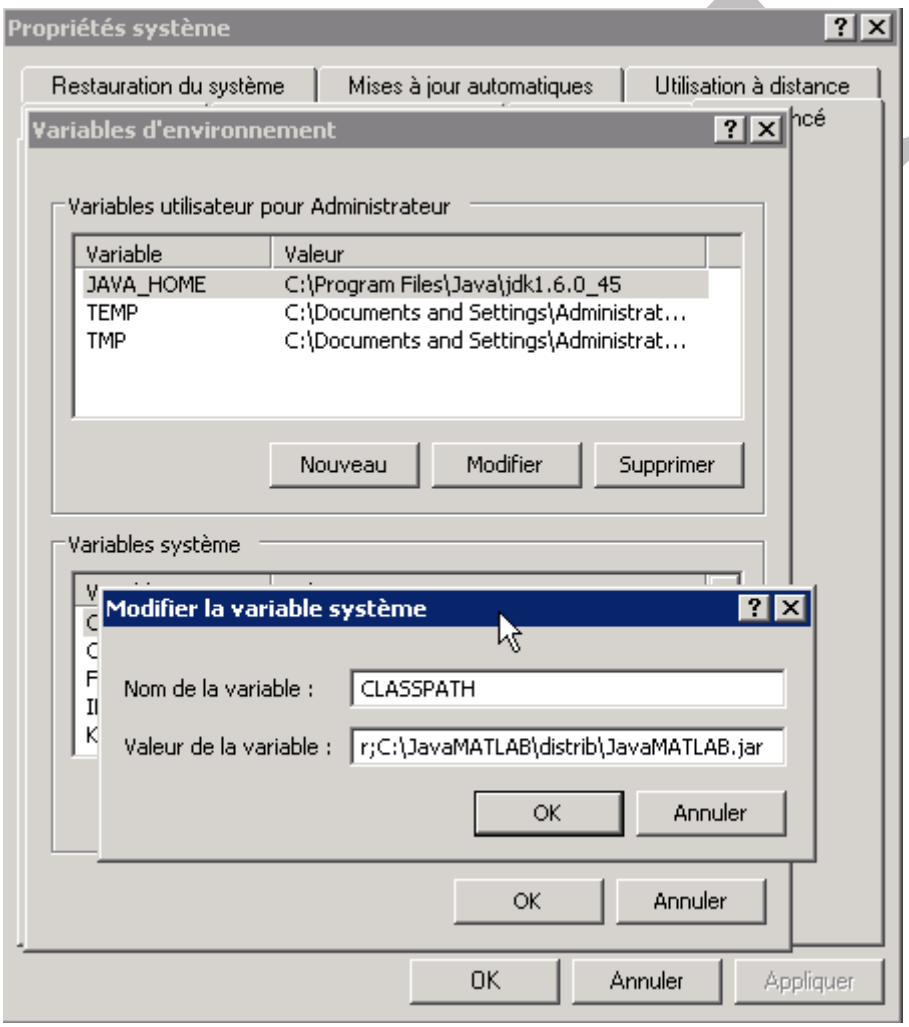
This will give you:



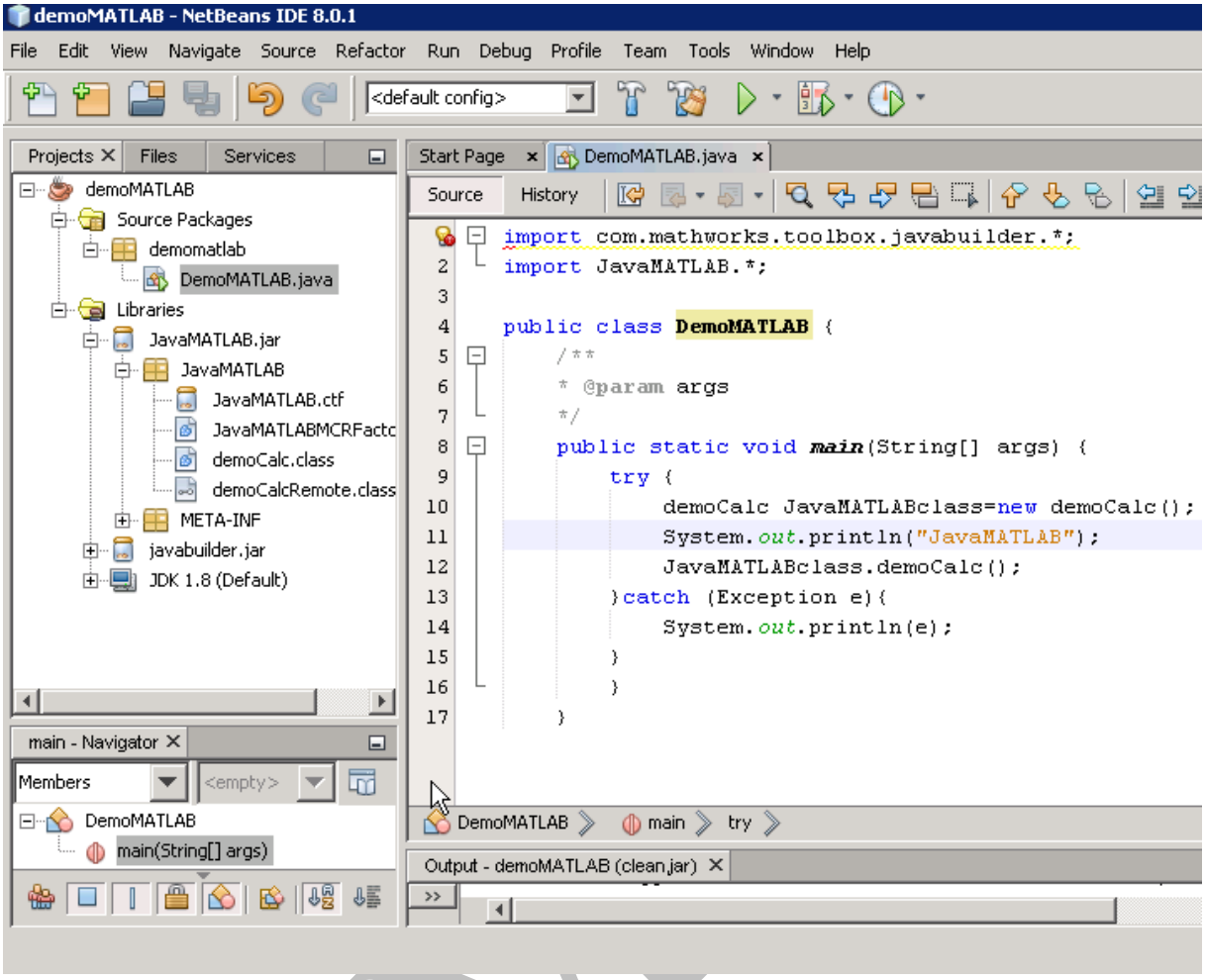
On some computers it is also necessary to add the following files and path ton system variables:

C:\Program Files\MATLAB\R2013a\toolbox\javabuilder\jar\javabuilder.jar

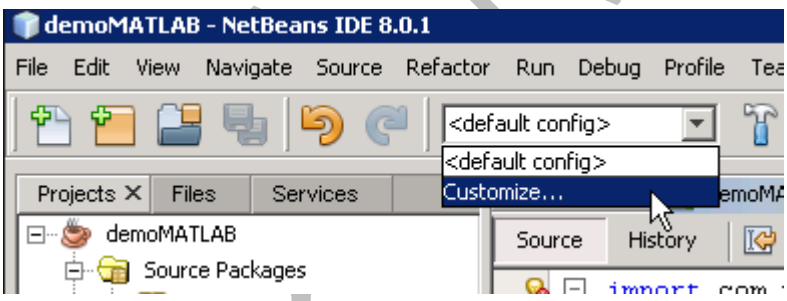
C:\JavaMATLAB\distrib\JavaMATLAB.jar

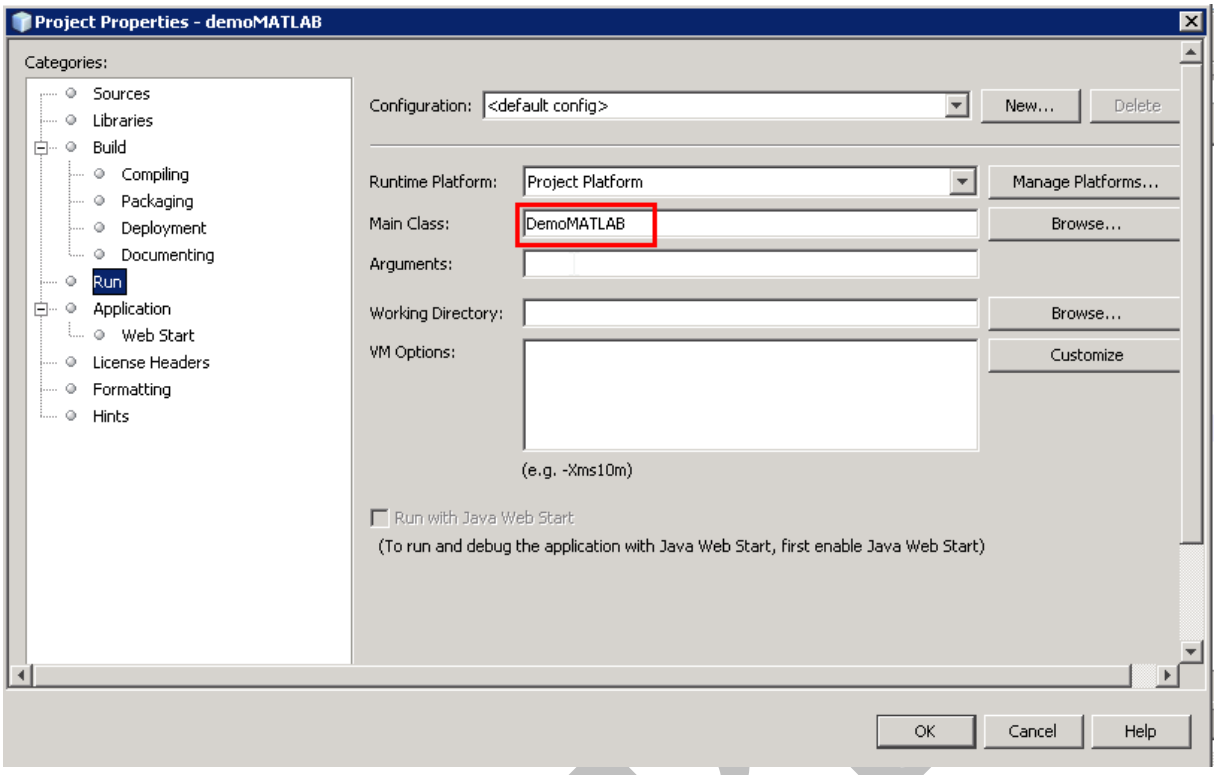


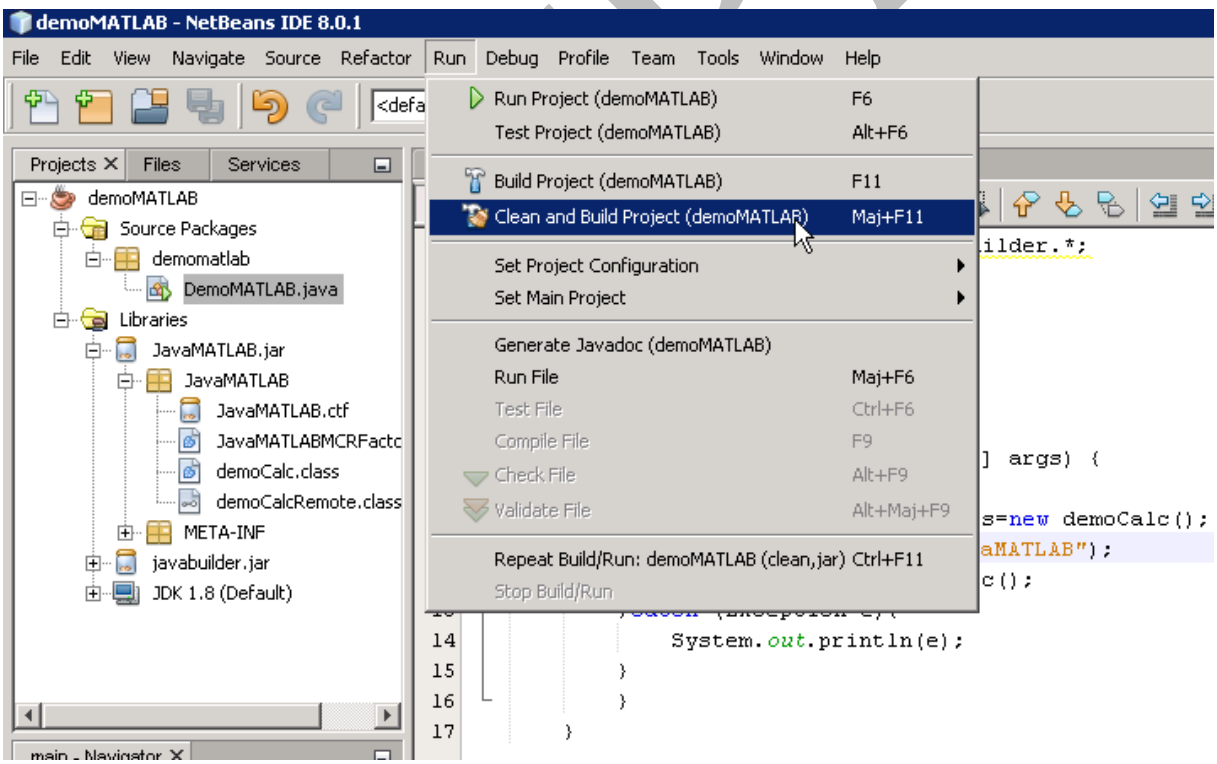
Once this is done don't forget to **restart you computer** (sometimes it helps to avoid issues during compilation).

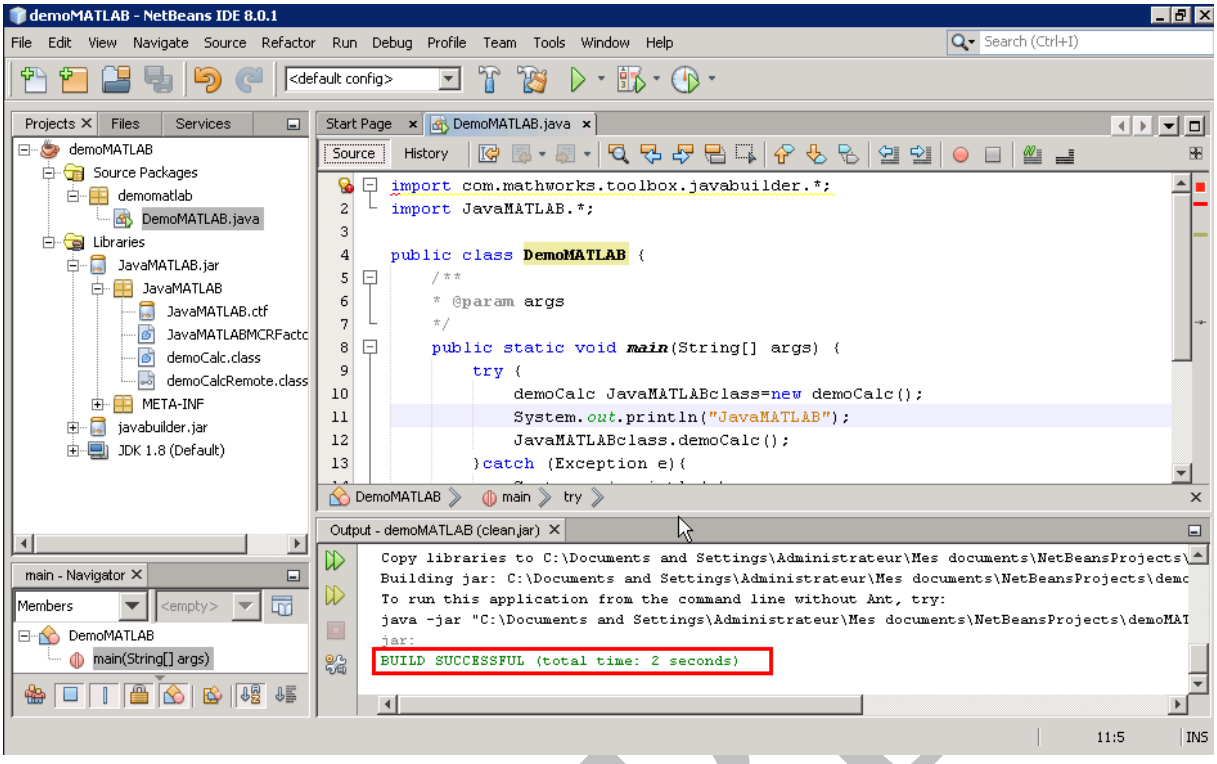


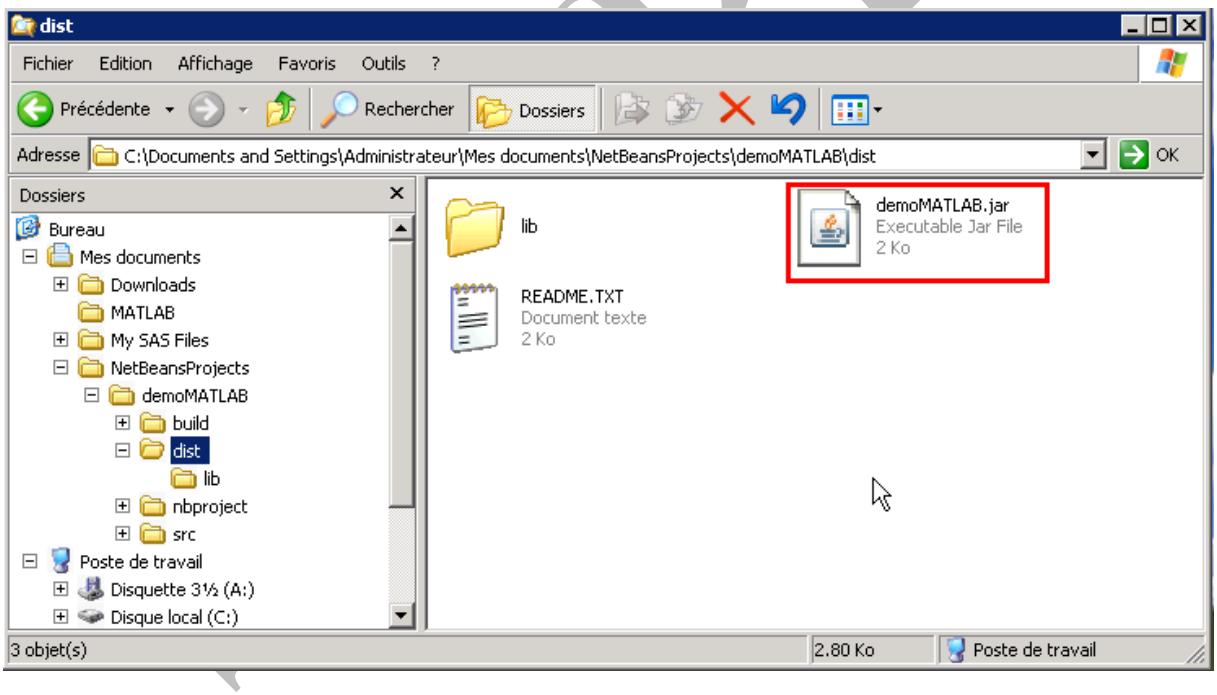
Beforing building the project check that the class name is really correct:



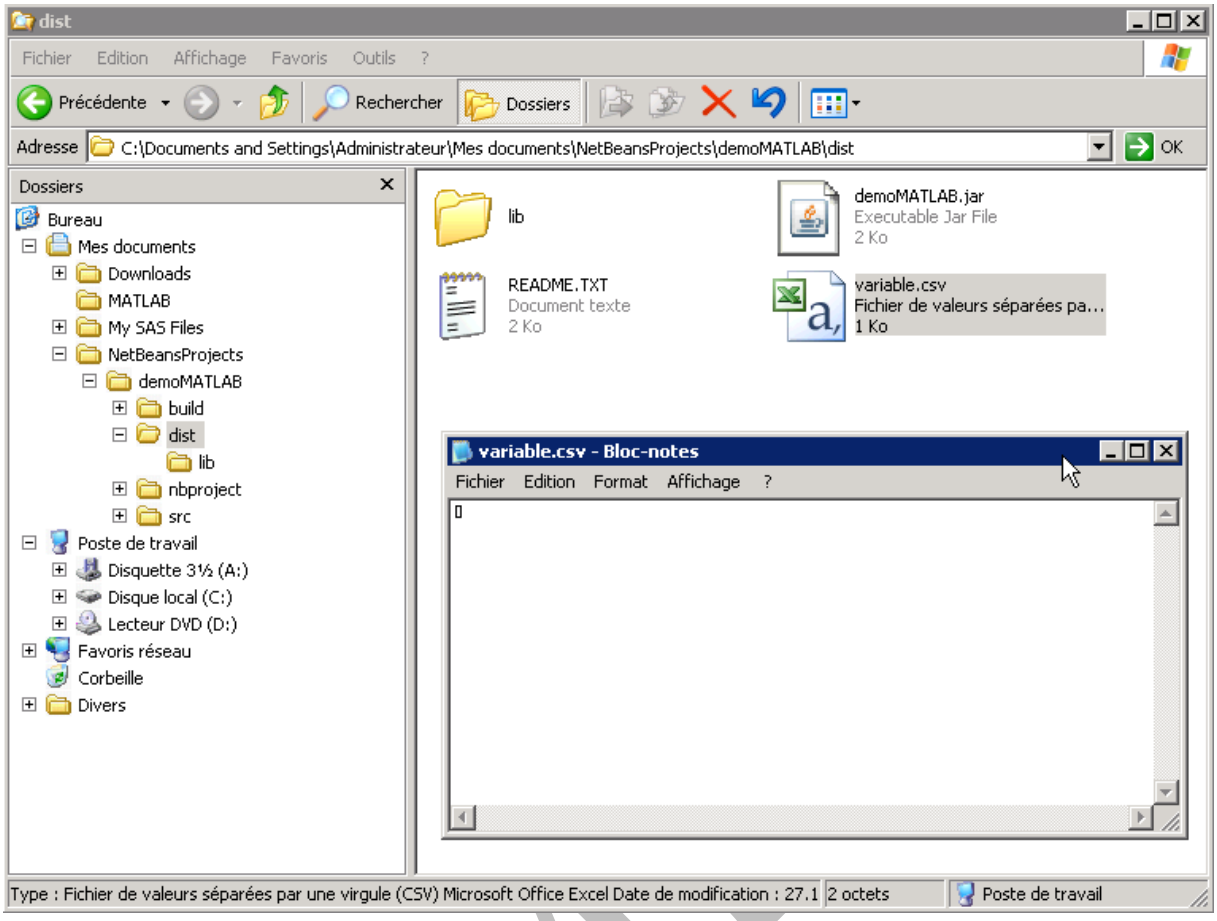

Project Properties - demoMATLAB
Categories:
Sources
Libraries
Build
Compiling
Packaging
Deployment
Documenting
Run
Application
Web Start
License Headers
Formatting
Hints
Configuration: <default config>
New...
Delete
Runtime Platform: Project Platform
Manage Platforms...
Main Class: DemoMATLAB
Browse...
Arguments:
Working Directory:
Browse...
VM Options:
Customize
(e.g. -Xms10m)
Run with Java Web Start
(To run and debug the application with Java Web Start, first enable Java Web Start)
OK
Cancel
Help


demoMATLAB - NetBeans IDE 8.0.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Run Project (demoMATLAB) F6
Test Project (demoMATLAB) Alt+F6
Build Project (demoMATLAB) F11
Clean and Build Project (demoMATLAB) Maj+F11
Set Project Configuration
Set Main Project
Generate Javadoc (demoMATLAB)
Run File Maj+F6
Test File Ctrl+F6
Compile File F9
Check File Alt+F9
Validate File Alt+Maj+F9
Repeat Build/Run: demoMATLAB (clean,jar) Ctrl+F11
Stop Build/Run
Projects
Files
Services
demoMATLAB
Source Packages
demomatlab
DemoMATLAB.java
Libraries
JavaMATLAB.jar
JavaMATLAB
JavaMATLAB.ctf
demoCalc.class
demoCalcRemote.class
META-INF
javabuilder.jar
JDK 1.8 (Default)
System.out.println(e);



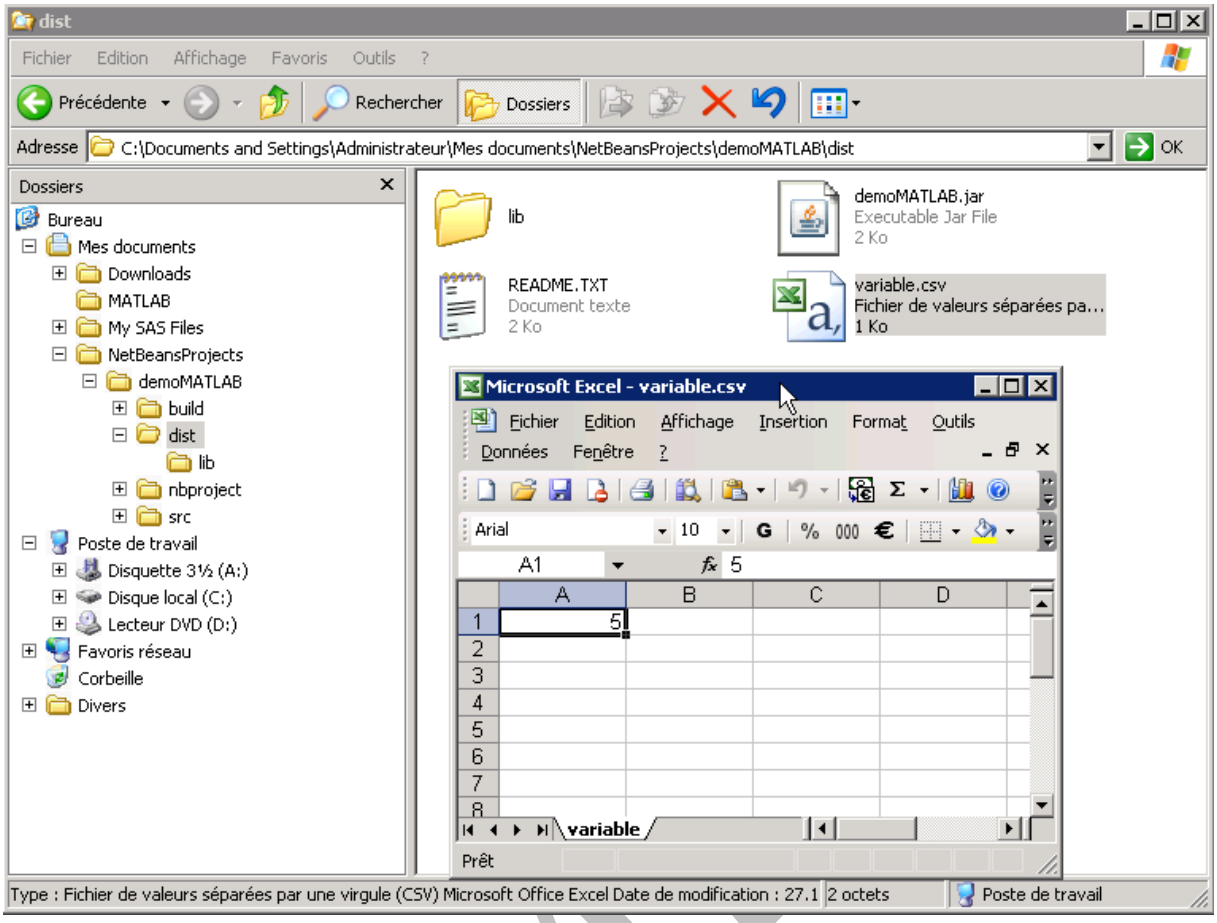


We get the expected result but when we open the file in the NotePad:



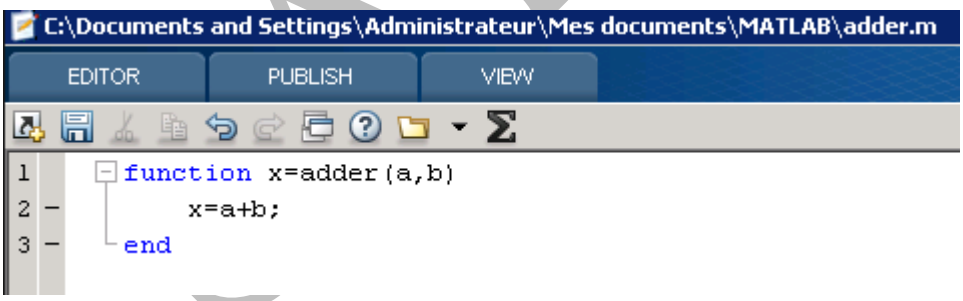
... And if we open it in Microsoft Excel:

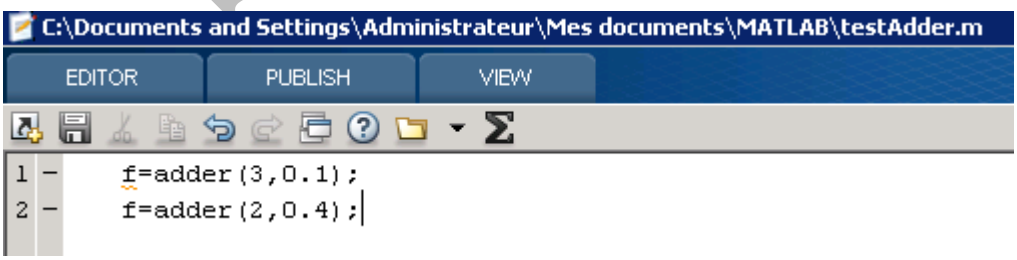
SAMPLE

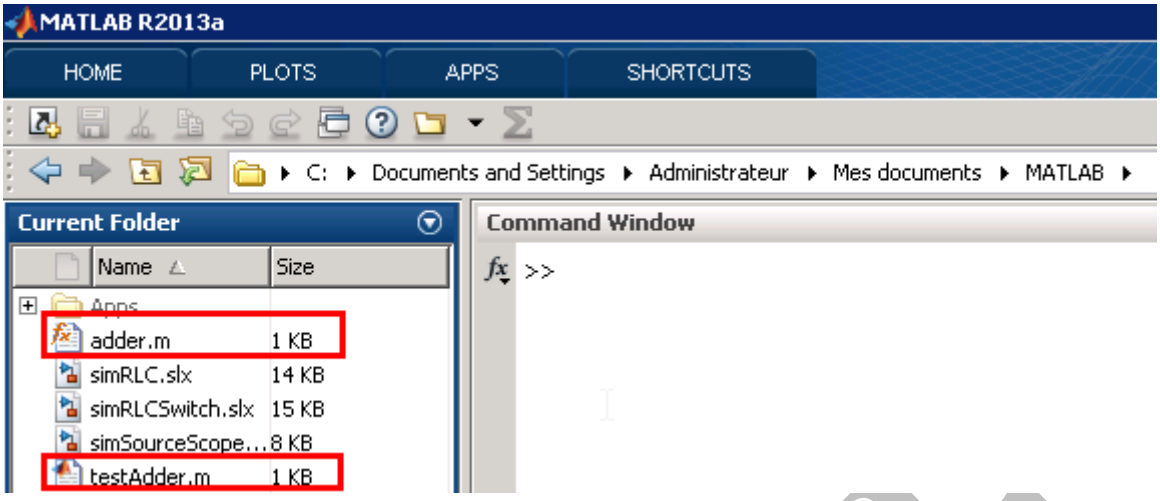


VHDL

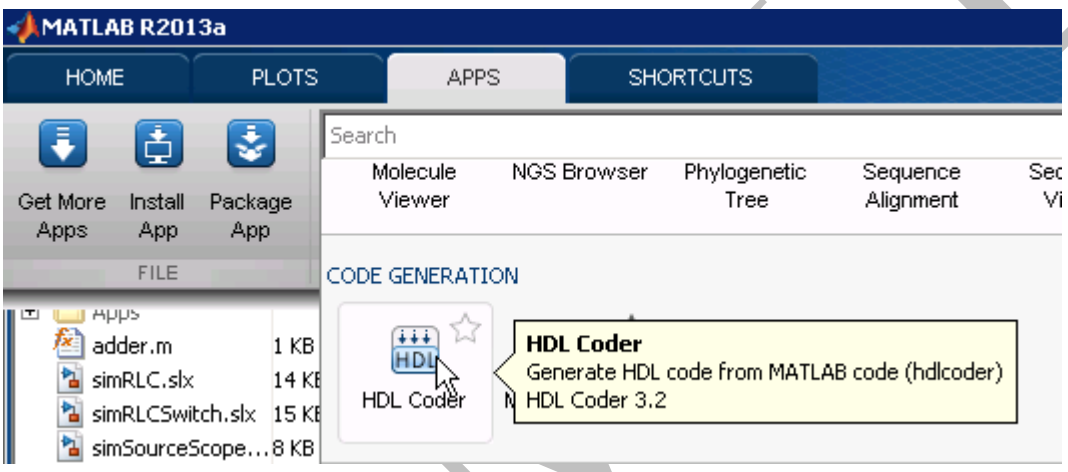
We will use two scripts for the topic. One containing the script and one to call it for test purposes.

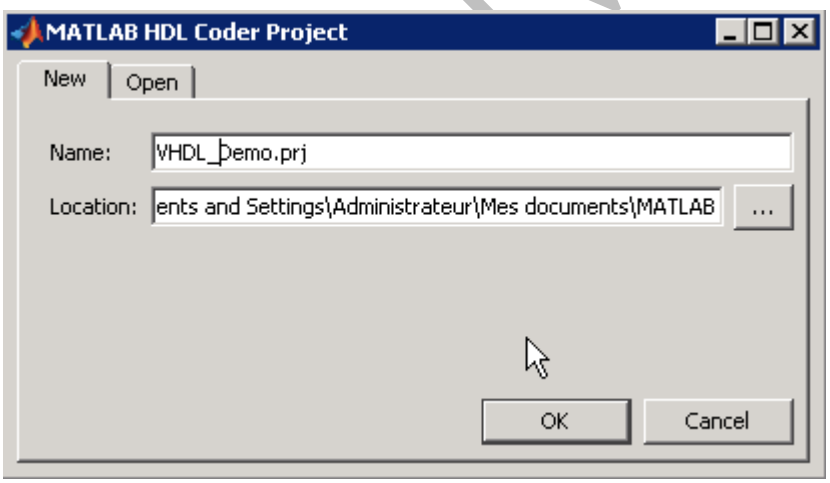


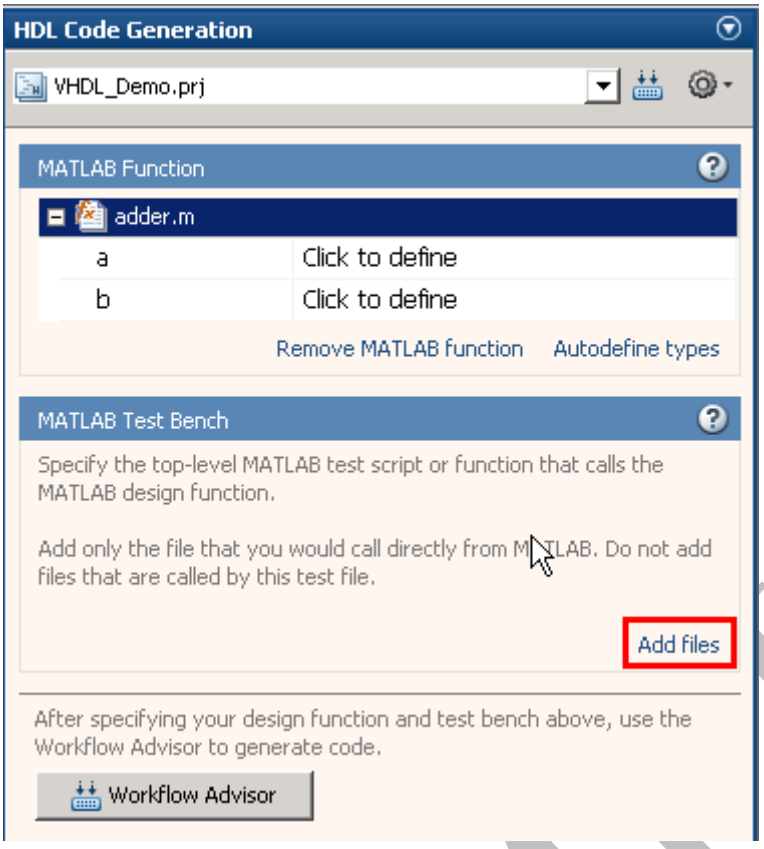


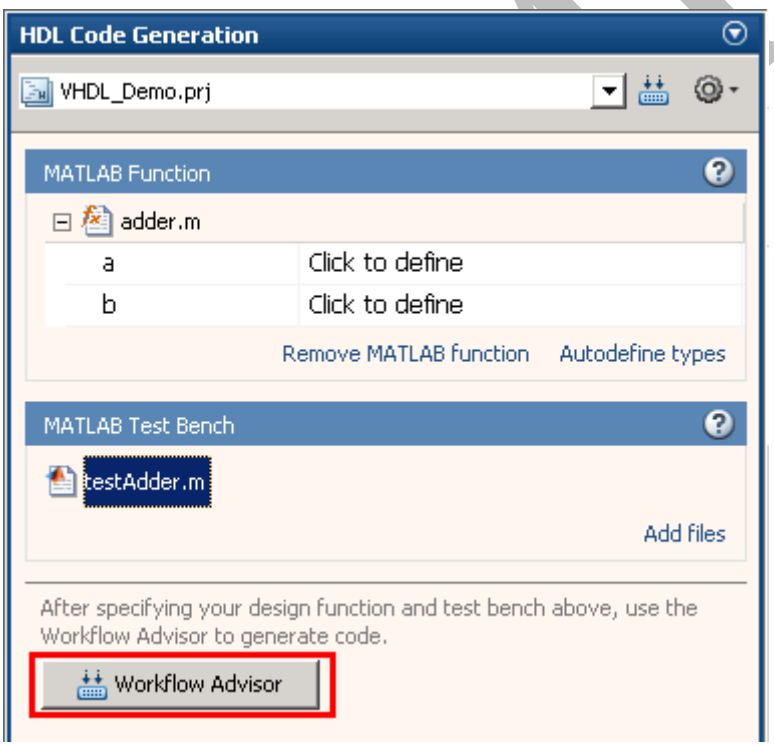


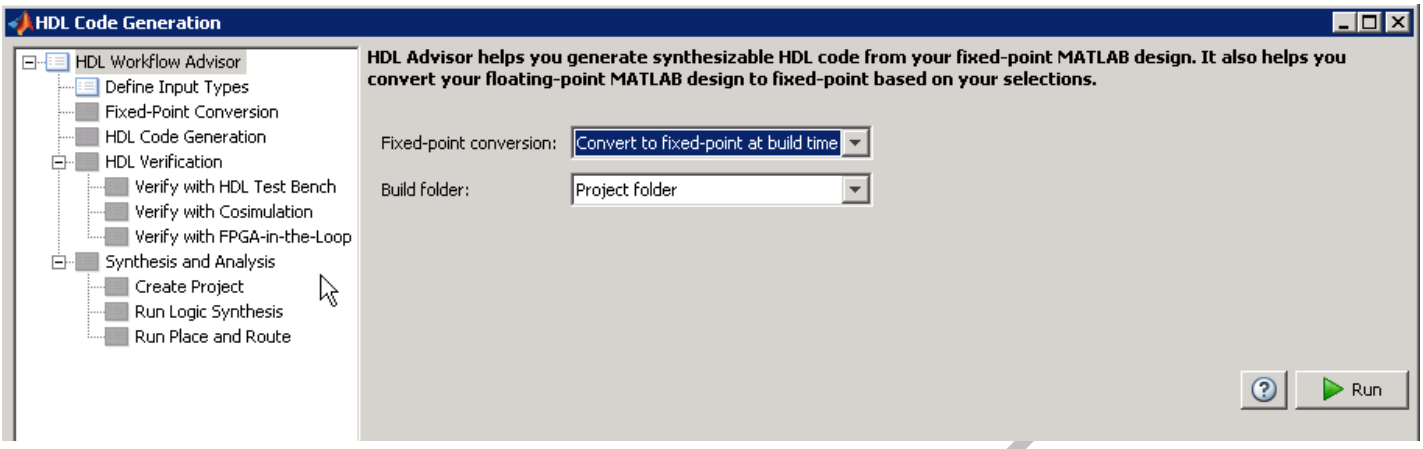
A very important thing: the active directory must have no spaces!!!

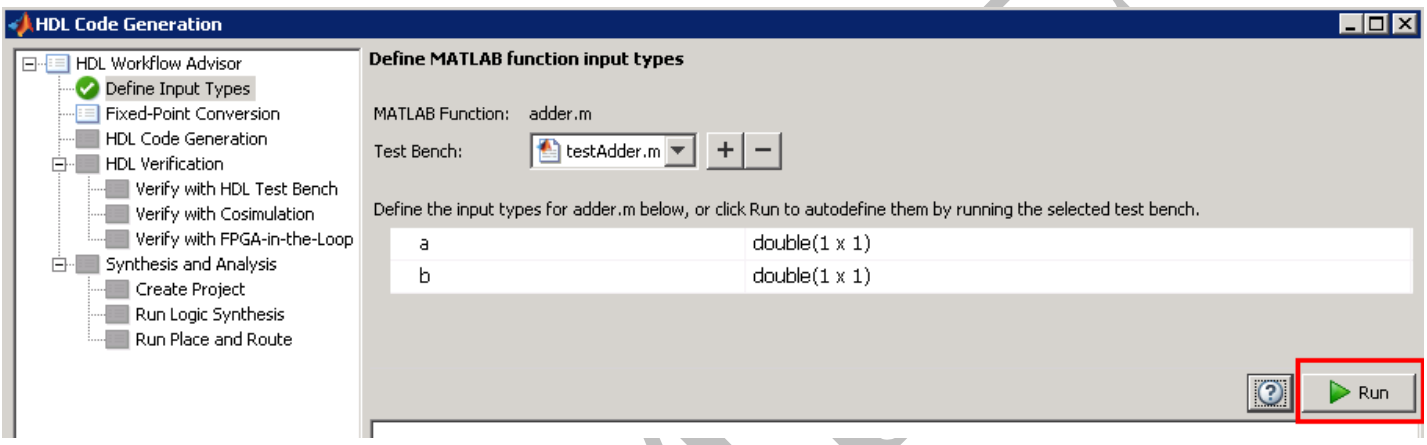


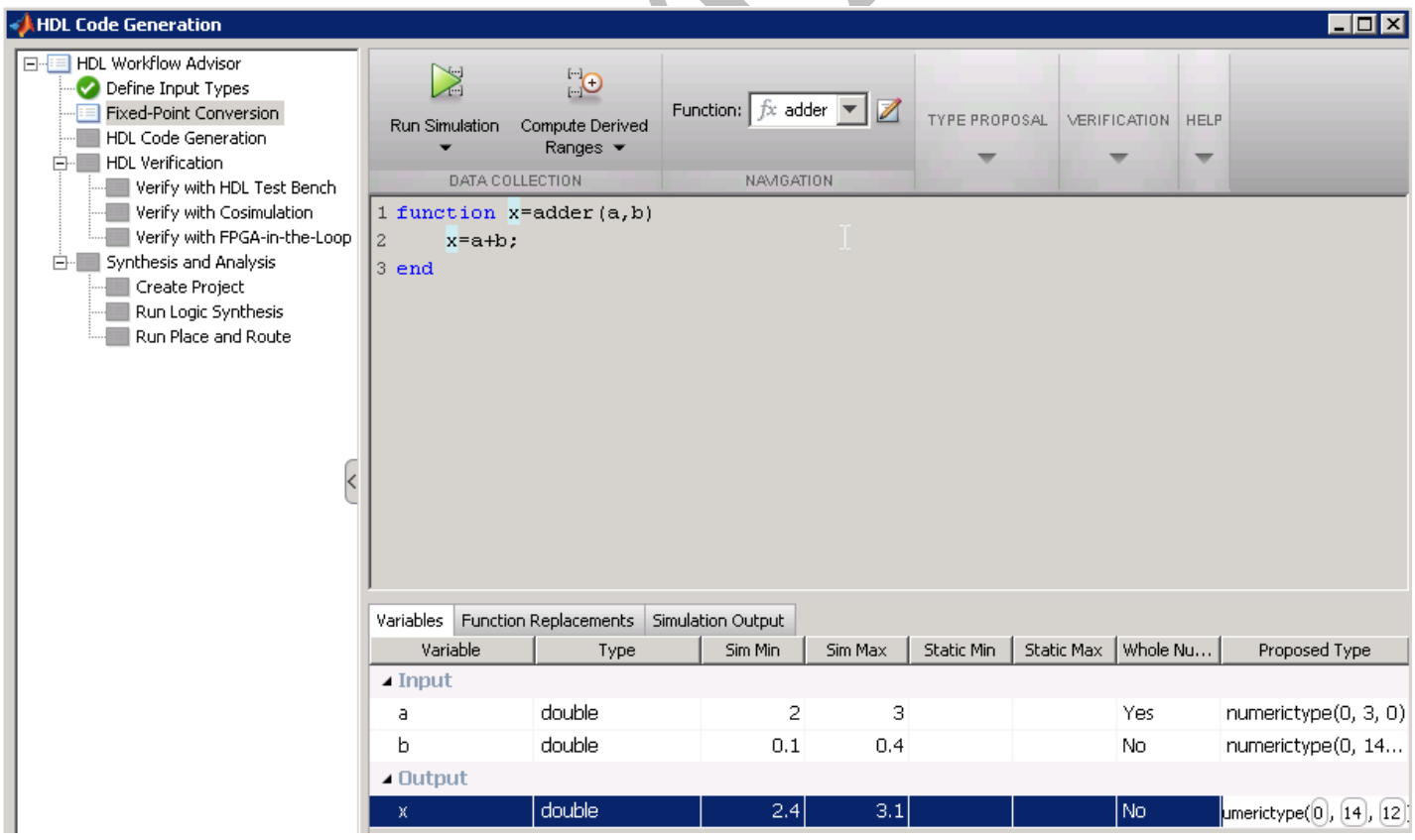



HDL Code Generation
VHDL_Demo.prj
MATLAB Function
adder.m
a
Click to define
b
Click to define
Remove MATLAB function
Autodefine types
MATLAB Test Bench
Specify the top-level MATLAB test script or function that calls the MATLAB design function.
Add only the file that you would call directly from MATLAB. Do not add files that are called by this test file.
Add files
After specifying your design function and test bench above, use the Workflow Advisor to generate code.
Workflow Advisor

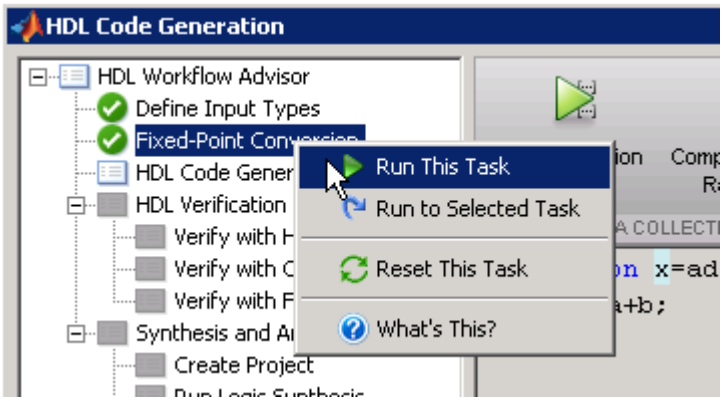

HDL Code Generation
VHDL_Demo.prj
MATLAB Function
adder.m
a
Click to define
b
Click to define
Remove MATLAB function
Autodefine types
MATLAB Test Bench
testAdder.m
Add files
After specifying your design function and test bench above, use the Workflow Advisor to generate code.
Workflow Advisor

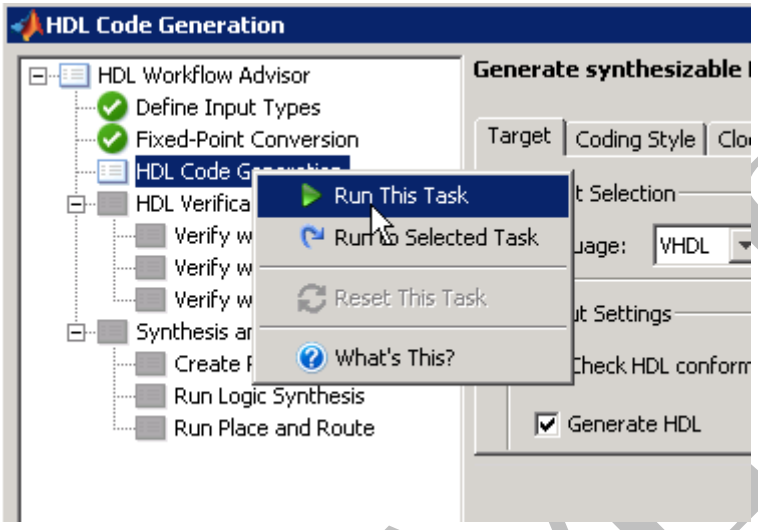


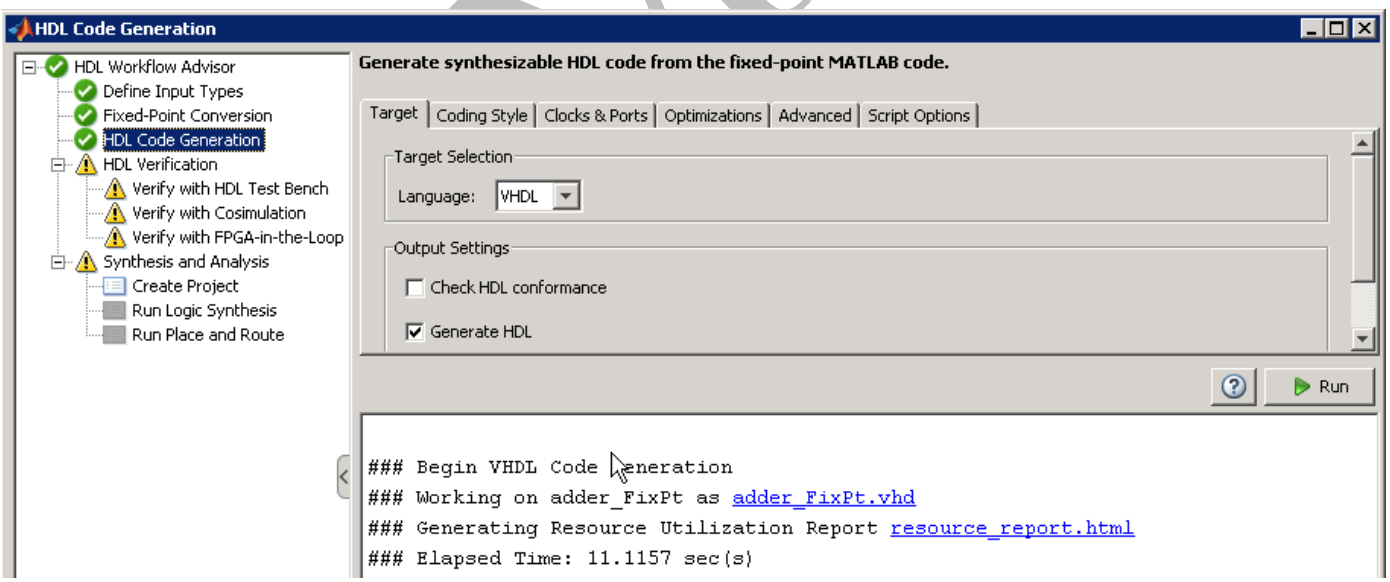




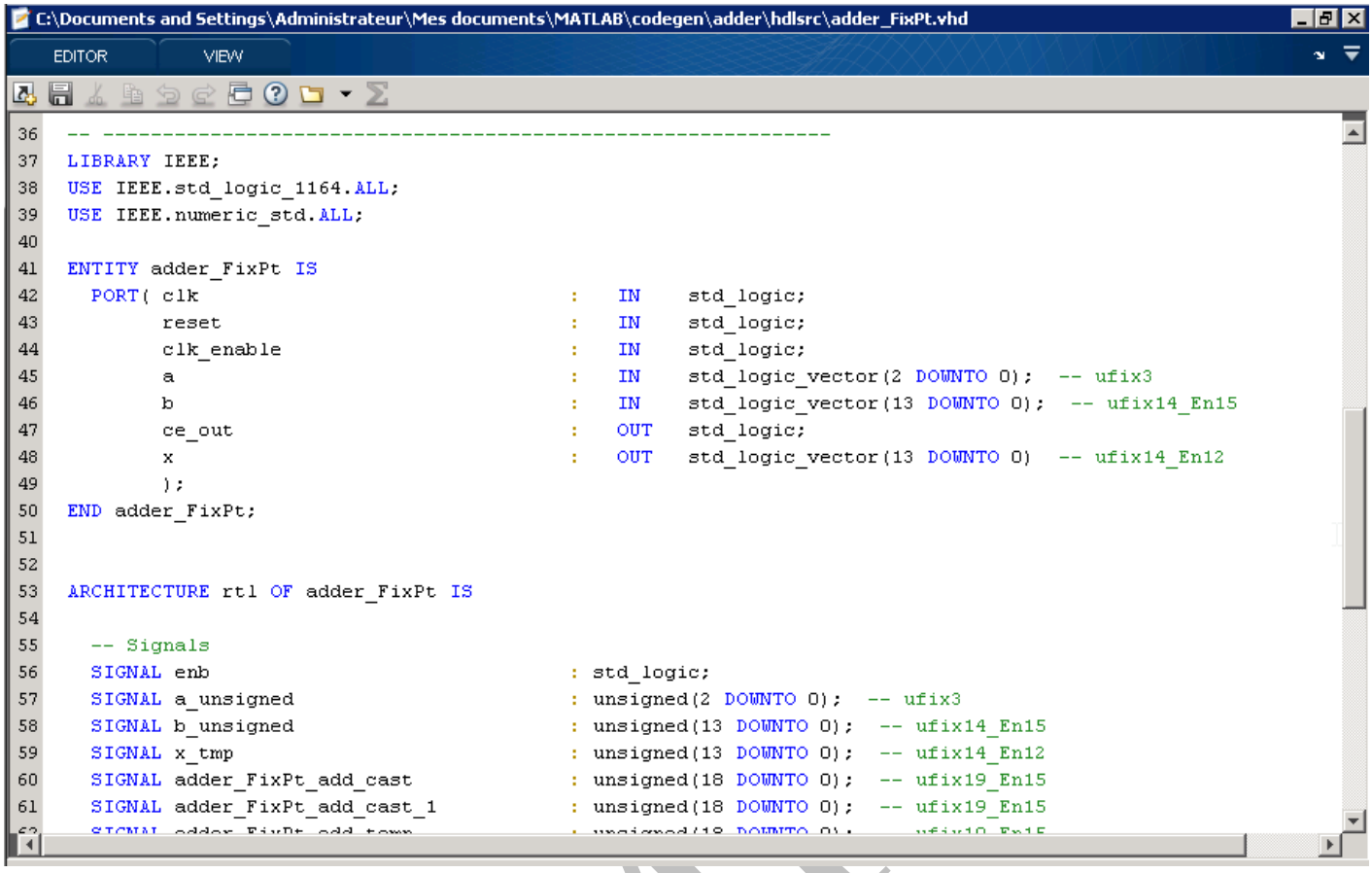
The best thing to do is to get the ✅ at each workflow step:







Have a look to the final code by clicking on the link adder_FixPt.vhd:


C:\Documents and Settings\Administrateur\Mes documents\MATLAB\codegen\adder\hdlsrc\adder_FixPt.vhd

EDITOR VIEW

```
36  -- -------------------------------------------------------------
37  LIBRARY IEEE;
38  USE IEEE.std_logic_1164.ALL;
39  USE IEEE.numeric_std.ALL;
40
41  ENTITY adder_FixPt IS
42    PORT( clk                               :   IN    std_logic;
43          reset                             :   IN    std_logic;
44          clk_enable                        :   IN    std_logic;
45          a                                 :   IN    std_logic_vector(2 DOWNTO 0);  -- ufix3
46          b                                 :   IN    std_logic_vector(13 DOWNTO 0);  -- ufix14_En15
47          ce_out                            :   OUT   std_logic;
48          x                                 :   OUT   std_logic_vector(13 DOWNTO 0)  -- ufix14_En12
49          );
50  END adder_FixPt;
51
52
53  ARCHITECTURE rtl OF adder_FixPt IS
54
55    -- Signals
56    SIGNAL enb                              : std_logic;
57    SIGNAL a_unsigned                       : unsigned(2 DOWNTO 0);  -- ufix3
58    SIGNAL b_unsigned                       : unsigned(13 DOWNTO 0);  -- ufix14_En15
59    SIGNAL x_tmp                            : unsigned(13 DOWNTO 0);  -- ufix14_En12
60    SIGNAL adder_FixPt_add_cast             : unsigned(18 DOWNTO 0);  -- ufix19_En15
61    SIGNAL adder_FixPt_add_cast_1           : unsigned(18 DOWNTO 0);  -- ufix19_En15
```

SAMPLE